ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Sistema de monitorización de constantes vitales con dispositivos inalámbricos

Alumno: Ibon Rodríguez Andrés

Tutor: Jesús Corres

Pamplona, 23 de Junio de 2014

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Agradecimientos

Quiero dedicar y agradecer este trabajo a mi familia, a mi madre, mi hermano y especialmente a mi padre.

A mis amigos y compañeros de clase, sobre todo a Jakue López y a Víctor Sancha por su ayuda recibida y como no a mi compañero de batalla en esta experiencia Luis Pérez.

A los profesores de la Universidad de Pinar del Río que se implicaron en el desarrollo de nuestro proyecto, sobre todo agradecer su ayuda a Jose Raúl Vento.

A los tutores, Yohany Rodríguez y Jesús Corres.

También agradecer a la Universidad de Pinar del Río, a la Universidad Pública de Navarra y al programa de Cooperación Internacional al Desarrollo la oportunidad que nos dieron de realizar este proyecto en Cuba.

Y por supuesto agradecer su ayuda a William Cabrera ya que sin él esto no hubiera sido posible.

A todos aquellos que han estado conmigo, a los que me han ayudado y apoyado tanto aquí como en Cuba, Muchas Gracias.

Resumen

El objetivo de este proyecto es realizar una aplicación Android que establezca una comunicación inalámbrica bluetooth entre un dispositivo móvil o un mini PC con sistema operativo Android y Arduino MEGA 2560 al que se le incorpora un sensor de humedad de fibra óptica para aplicaciones biomédicas.

Para ello, la aplicación establecerá la conexión inalámbrica bluetooth entre el dispositivo móvil y Arduino MEGA 2560 mediante el módulo bluetooth LM780.
Una vez abierta la aplicación y establecida la conexión bluetooth, esta empieza a recibir los datos de los sensores. Los valores se reflejan tanto en siete barras progresivas correspondientes a cada uno de los sensores como numéricamente a la derecha de cada barra progresiva. Si pulsamos el botón *GRÁFICAS* en la pantalla principal nos lleva a otra pantalla en la que se muestra una lista de elementos en la cual cada elemento da la posibilidad de acceder a la gráfica en tiempo real de su variable correspondiente. Una vez elegido el sensor deseado seleccionamos el elemento correspondiente a dicho sensor y pasamos a la siguiente pantalla. Esta muestra la gráfica en tiempo real de los valores del sensor correspondiente. En la gráfica se recoge una muestra por segundo.

El proyecto se ha realizado en un total de 7 fases con una duración aproximada de 145 días. El presupuesto del proyecto entre materiales y mano de obra asciende a un total de 4441.05 €, IVA incluido.

La recepción de comandos desde Arduino al dispositivo móvil se ha conseguido desarrollar satisfactoriamente, ya que se han alcanzado todos los objetivos propuestos. En cuanto al envío de comandos desde el dispositivo móvil a Arduino, no se ha conseguido alcanzar todos los objetivos propuestos inicialmente, sobre todo por falta de tiempo y recursos, lo cual se pretende continuar en futuras líneas de investigación.

1. Introducción y objetivos

1.1. Introducción

El vínculo que forman la tecnología y la salud cada vez es mayor, gracias a esto, el cuidado de la salud y la cura de enfermedades ha ido en aumento. Dada la demanda que se esta originando de atención médica y la falta de espacios en los hospitales, la atención a la salud ha dado origen a un nuevo concepto denominado monitoreo externo o remoto. Este concepto trata básicamente la implementación de una serie de dispositivos que reportan en tiempo real la situación de los signos vitales de un paciente. El monitoreo remoto beneficia tanto al paciente como al médico, ya que, por medio de un equipo de monitoreo y el uso de las tecnologías emergentes, se permite una comunicación a distancia.

Este proyecto nace con la necesidad de realizar un sistema de monitorización de constantes vitales con dispositivos inalámbricos, más concretamente mediante la tecnología inalámbrica bluetooth. El objetivo de este proyecto consiste en realizar la conexión inalámbrica bluetooth entre un dispositivo móvil con sistema operativo Android y Arduino Mega 2560.

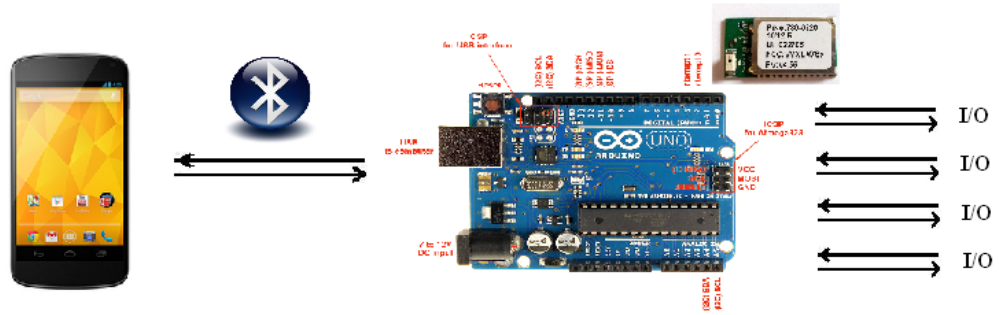


Figura 1.1: Arquitectura del sistema

El objetivo principal de este proyecto no solo consiste en realizar la conexión inalámbrica bluetooth entre un dispositivo móvil y Arduino sino también familiarizarse con el entorno de trabajo Eclipse para el desarrollo de aplicaciones móviles Android. La ventaja del sistema operativo Android tanto para usuarios como para desarrolladores es

que ha sido distribuido como código abierto a diferencia de iOS de iPhone. Lo que ha supuesto que el crecimiento de aplicaciones disponibles sea constante. Gracias a esto la información entre desarrolladores tanto nóveles como expertos es muy grande y supone una ayuda indispensable para comprender su funcionamiento en conjunto y para desarrollar una nueva aplicación.

La implementación de la aplicación está orientada a ser probada con el terminal LG-E960 Nexus 4 con sistema operativo Android con versión Android 4.3 (Jelly Bean).

La parte de Introducción a Arduino, montaje de la placa y programación con Arduino ha sido un proyecto en parte conjunto realizado en la Universidad de Pinar del Río, Cuba.

1.2. Objetivos

El objetivo de este proyecto consiste en realizar la conexión inalámbrica Bluetooth entre un dispositivo (Arduino MEGA 2560), al que se le incorpora un sensor de humedad de fibra óptica para aplicaciones biomédicas y una aplicación Android en el móvil o en un mini PC con sistema operativo Android.

El sensor de fibra óptica debe de mostrar las siguientes señales:

1. SENSOR_FO 1
2. SENSOR_FO 2
3. TEMP_REF_1
4. TEMP_REF_2
5. TEMP_INTERNA
6. RH_REF_1
7. RH_REF_2

Todas se transmitirán con 32 bits en coma flotante, ieee 754.
La frecuencia máxima de refresco será de 0.1 Hz.
La aplicación debe de tener dos modos de funcionamiento:

Modo 1
Monitorización: Cada segundo actualiza los datos y permite enviar comandos.

Modo 2
Permite seleccionar una o varias variables y muestra un gráfico con los X últimos valores. El escalado y otros factores de la presentación de la gráfica se decidirán durante el desarrollo del proyecto.

La aplicación deberá enviar los siguientes comandos de control a la máquina:

1. CONFIG_vars 0xXXXX Configuración de las variables muestreadas
2. CONFIG_frec 0xXXXX Configuración del tiempo de refresco
3. CONFIG_on 0xXXXX ON/OFF envío de datos

La aplicación recibirá los datos en el siguiente formato:
DATA v1 0.123 v2 6.34 v3 30.443 v4 30.123 v5 46.34 v6 45.443 v7 54.44

2. Placa de montaje y entorno de programación Arduino

2.1. Introducción a Arduino

Tenemos la necesidad de disponer de un sistema capaz de procesar y trabajar con la información, así como de dirigirla, que hace que sea necesario emplear un sistema hardware específico, capaz de leer señales analógicas, escribir señales digitales y transmitir dicha información a otros dispositivos de una manera determinada por nosotros. En nuestro caso hemos optado por Arduino.

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se desarrolló para cualquier interesado en crear entornos u objetos interactivos.

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino, muy intuitivo y similar a C++ y que consta de una gran variedad de funciones ya implementadas, así como permite la introducción de librerías con otras nuevas. Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software.

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta.

Este hecho permite que mucha gente ya trabaje bajo esta plataforma por lo que es bastante sencillo encontrar referencias en internet.

En nuestro proyecto optamos por una placa Arduino Mega 2560 y un módulo bluetooth LM780, cuyas características técnicas se adjuntan en los **ANEXOS I** y **II** respectivamente. Dichos Anexos se añaden para facilitar la información de futuros proyectos relaccionados en la Universidad de Pinar del Río, Cuba.

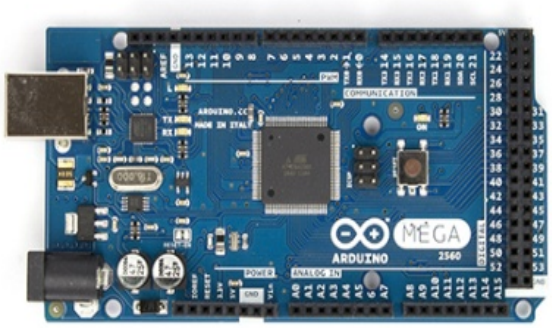


Figura 2.1: Placa Arduino Mega 2560

Figura 2.2: Módulo bluetooth LM780

Arduino Mega 2560 es un microcontrolador basado en el procesador ATmega2560 que consta de 54 pines de entradas/salidas digitales, 16 entradas analógicas, 4 puertos UART (puertos de serie hardware), un oscilador de 16 MHz, conexión USB, conexión para alimentación externa y un botón de reset.

2.2. Montaje de la placa

Los componentes que vamos a utilizar en el montaje de la placa son los siguientes:

- 1 Arduino MEGA 2560
- 1 Módulo Bluetooth LM-780
- 8 Resistencias de 6.8 K para la lectura y simulación de variables
- 3 Resistencias de 180 K para los diodos LED
- 1 Resistencia de 100 K para el pulsador
- 3 diodos LED rojos
- 1 Pulsador
- 1 Potenciómetro de 20 K
- 2 Condensadores cerámicos de 0.1 µF
- Conectores de paso
- Placa de impresión PCB, cable de un único hilo de diferentes colores
- Cinta termo retráctil

El esquema del circuito diseñado para el montaje y la placa son los siguientes:

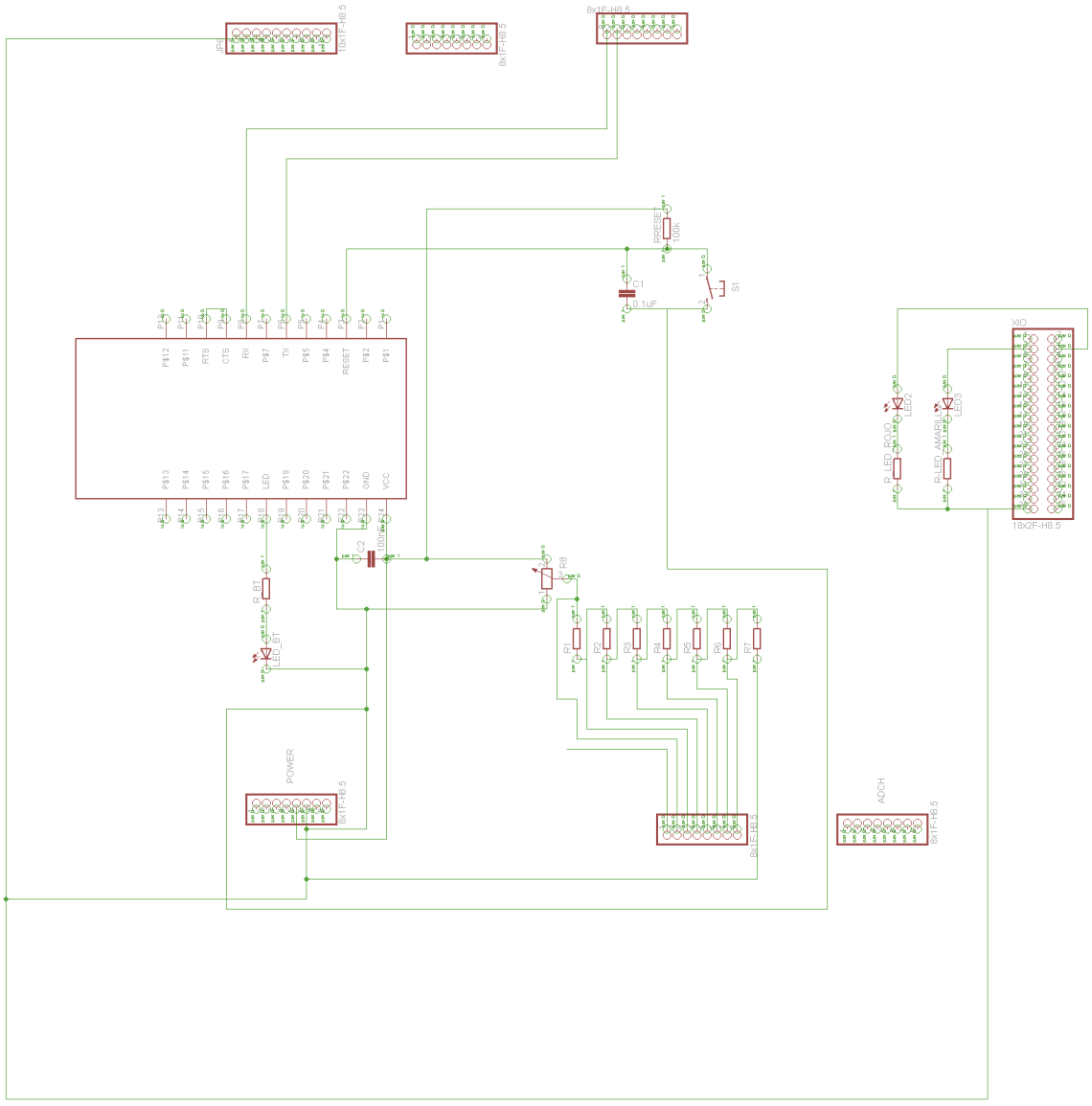


Figura 2.3: Esquema del circuito

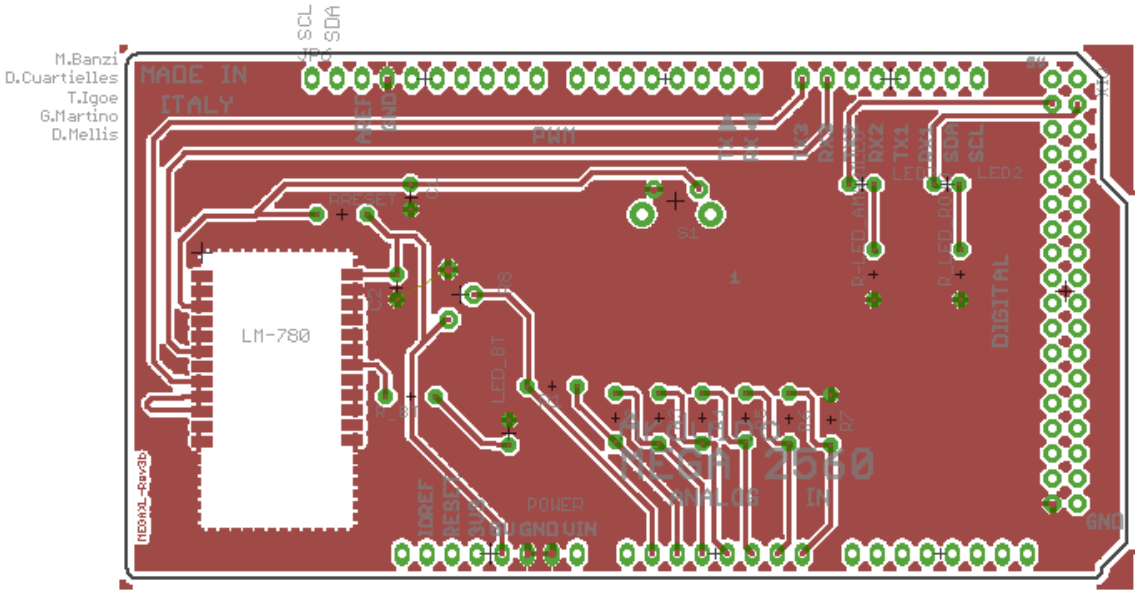


Figura 2.4: Placa del circutio

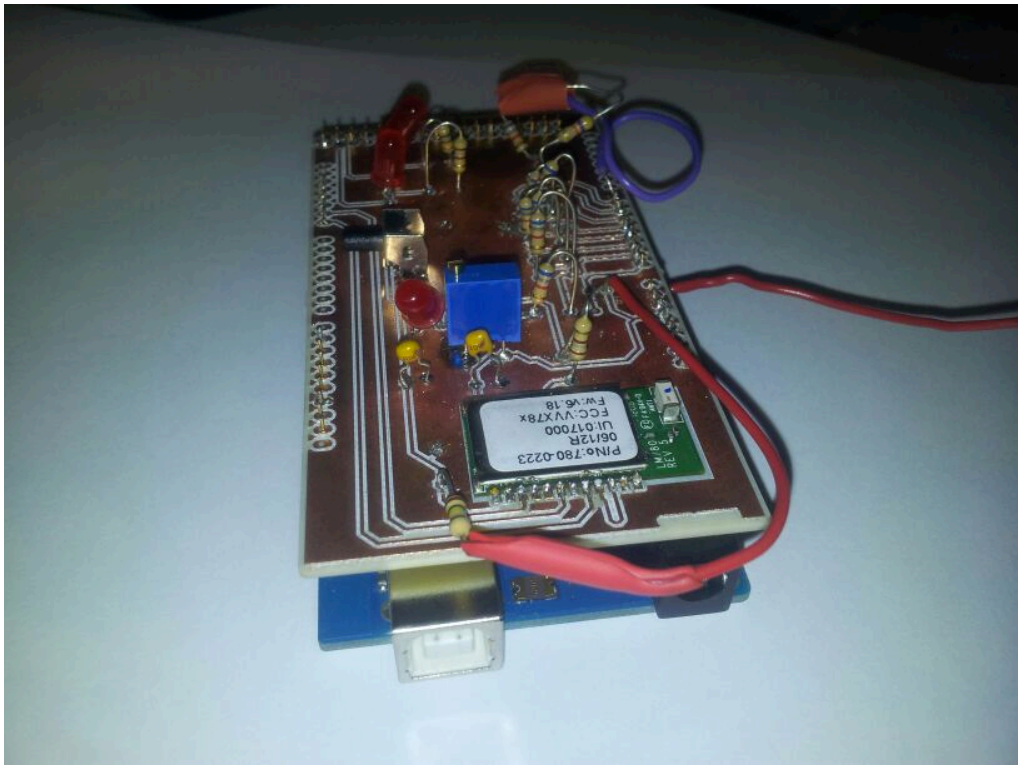


Figura 2.5: Imagen de la placa del circutio

2.3 Programación en Arduino

Programa en el que se leen 7 señales DC constantes que se pueden modificar por medio de un potenciómetro. Se enciende el LED situado en el PIN Digital 23.

Se transmite conforme al protocolo establecido de la siguiente manera y en este caso por el puerto serie 3 a 19200 bps.

Protocolo establecido:
<Longitud D0, Datos 0, Longitud D1, Datos 1, Longitud D2, Datos 2, Longitud D3, Datos 3, Longitud D4, Datos 4, Longitud D5, Datos 5, Longitud D6, Datos 6, Longitud D7, Datos 7>

Programa

```
/* LecturaVariables

 <Longitud D0, Datos 0, Longitud D1, Datos 1, ..... , Longitud D7, Datos 7>

 */


 byte cuentaChars(unsigned long N){

 /*FUNCIÓN QUE CUENTAL EL NÚMERO DE CARACTERES QUE TIENE UN NÚMERO ENTERO DE 2 BYTES CUYO VALOR MÁXIMO ES 2^16=65536, POR TANTO 5 CARACTERES=1byte*/

  if (N>=0 && N<10){ return(1); }

  if (N>=10 && N<100) { return (2); }

  if (N>=100 && N<1000) { return (3) ; }

  if (N>=1000 && N<10000) { return (4) ; }

  if (N>=10000 && N<100000) { return (5); }

 }


/*INICIALIZACIÓN DE VARIABLES*/

 char val;

 int sensorValue = 0;

 int valor1 = 0;
```

```
 int valor2 = 0;
 int valor3 = 0;
 int valor4 = 0;
 int valor5 = 0;
 int valor6 = 0;
 int valor7 = 0;

int PowerLED = 23;    //seleción del PIN con el LED ROJO (Power);
int LED_Amarillo = 22;   //selcción del PIN con el LED AMARILLO (Control)

void setup() {
 // Inicialización del puerto Serie3 donde se encuentra el transmisor BT
 Serial3.begin(19200);
 // Configuración de los pines digitales (LEDs)
 pinMode(PowerLED, OUTPUT);
 pinMode(LED_Amarillo, OUTPUT);
 digitalWrite(PowerLED, HIGH); //Siempre encendido el LED ROJO

}

void loop() {
delay(500);
//Lectura y envio de datos si el puerto serie está disponible
if (Serial3.available()) {
```

```
sensorValue = analogRead(A0);
valor1 = analogRead(A1);
valor2 = analogRead(A2);
valor3 = analogRead(A3);
valor4 = analogRead(A4);
valor5 = analogRead(A5);
valor6 = analogRead(A6);
valor7 = analogRead(A7);

Serial3.print('<');
Serial3.print(cuentaChars(sensorValue)); Serial3.print(sensorValue);delay(1);
Serial3.print(cuentaChars(valor1)); Serial3.print(valor1);delay(1);
Serial3.print(cuentaChars(valor2)); Serial3.print(valor2);delay(1);
Serial3.print(cuentaChars(valor3)); Serial3.print(valor3);delay(1);
Serial3.print(cuentaChars(valor4)); Serial3.print(valor4);delay(1);
Serial3.print(cuentaChars(valor5)); Serial3.print(valor5);delay(1);
Serial3.print(cuentaChars(valor6));  Serial3.print(valor6);delay(1);
Serial3.print('>');
 }
 }
```

3. Entorno de programación Eclipse

3.1. Introducción a Android

Android es un sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, inicialmente desarrollado por Android, Inc. Google respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto a la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (*surface manager*), un framework OpenCore, una base de datos relacional SQLite, una Interfaz de programación de API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, SSL y una biblioteca estándar de C Bionic. El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2,8 millones de líneas de lenguaje C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

La arquitectura del sistema operativo Android es el siguiente:

- **Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

- **Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- **Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

- **Runtime de Android**: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La

Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".

- **Núcleo Linux**: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

  Permite una sencilla reutilización de componentes y comunicación entre aplicaciones, siempre sujetas a ciertas medidas de seguridad, que facilita, por ejemplo, la actualización o sustitución de componentes por parte del usuario, resultando un sencillo y efectivo método para utilizar novedades o introducir mejoras en el software. Los principales conjuntos de servicios ofrecidos son los siguientes:

- Un extenso y variado conjunto de vistas (*Views*) ofrecidas para el diseño de interfaces gráficas de usuario y su interactividad con el sistema, como los típicos botones, cuadros de texto o listas.

- Los proveedores de contenidos (*Contents Providers*) son el método de intercambio de información entre aplicaciones, ya sea compartiendo los datos propios o accediendo a los de otras aplicaciones.

- Los gestores de recursos (*Resources Manager*) permiten el acceso indexado a recursos como cadenas o gráficos, en un intento de modular aún más el diseño de las aplicaciones y el uso de sus recursos.

- El gestor de notificaciones (*Notification Manager*) dirige alertas personalizadas al software, que son mostradas en una barra de estado.

- El gestor de actividades (*Activities Manager*) es responsable del ciclo de vida de las actividades. El ciclo de vida de las actividades no se trata únicamente de abrir y cerrar a gusto del usuario, si no que éstas, una vez iniciadas, permanecen cargadas en memoria siempre que se disponga de recursos para ello. En caso contrario el propio sistema operativo se encargará de destruirlas definitivamente. Dicho ciclo de vida se rige por las llamadas a los métodos *onCreate, onStart, onResume, onPause, onStop, onDestroy* y *onRestart.*

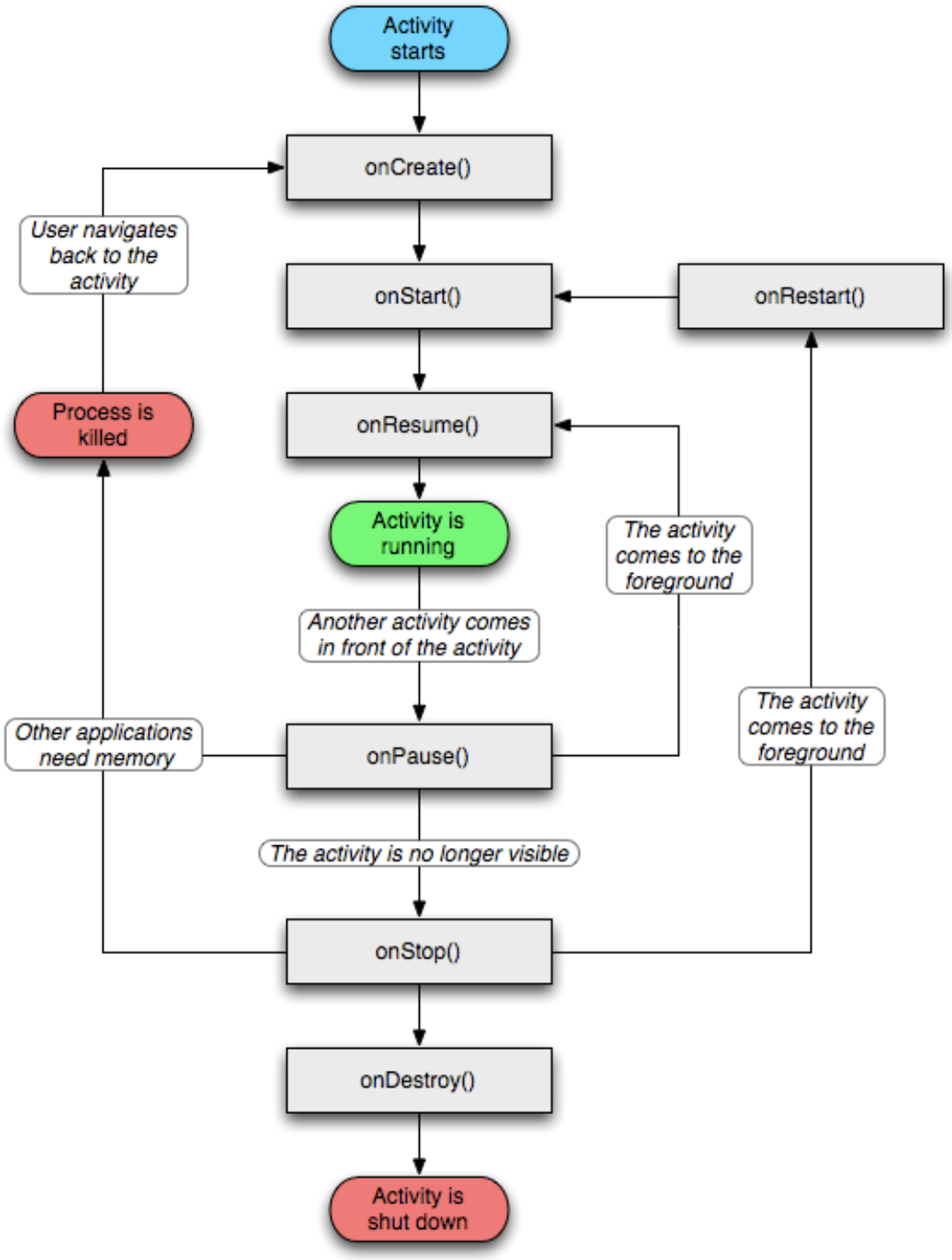


Figura 3.1: Ciclo de vida de una Activity.

3.2. Instalación del entorno de trabajo

Google ha preparado el paquete de software Android SDK, que incorpora todas las herramientas necesarias para el desarrollo de aplicaciones en Android. En él se incluye

conversor de código, debugger, librerías, emulador, documentación, ejemplos de código, etc. Todas estas herramientas son accesibles desde la línea de comandos, por otra parte para el desarrollo.
No obstante la mayoría de desarrolladores prefieren utilizar un IDE, o entorno de desarrollo integrado que integre un editor de texto con todas las herramientas de desarrollo. Aunque no son las únicas dos posibilidades, las alternativas más recomendables son Eclipse e IntelliJ Idea. Dado que es frecuente los problemas con el entorno de desarrollo, puede ser una buena idea instalar las dos y utilizar el que menos problemas nos de. A continuación pasamos a describir varias alternativas para el proceso de instalación del SDK Android.

Una instalación con IDE Eclipse requiere la instalación de los siguientes elementos:

- Java Runtime Environment 5.0 o superior.
- Eclipse (Eclipse IDE for Java Developers).
- Android SDK (Google).
- Eclipse Plug-in (Android Development Tools - ADT).

3.2.1. Instalación de la máquina virtual java

Este *software* va a permitir ejecutar código Java en tu equipo. A la máquina virtual Java también se la conoce como entorno de ejecución Java, Java Runtime Environment (JRE) o Java Virtual Machine (JVM).
Muy posiblemente ya tengas instalada la Máquina Virtual Java en tu equipo. Si es así puedes pasar directamente al punto siguiente. En caso de dudas, puedes pasar también al punto siguiente. Al concluirlo te indicará si la versión de la máquina virtual Java es incorrecta. En caso necesario, regresa a este punto para instalar una adecuada. Para instalar la Máquina Virtual Java accede a http://java.com/es /download/ y descarga e instala el fichero correspondiente a tu sistema operativo.

3.2.2. Instalación basada en Eclipse

Eclipse resulta el entorno de desarrollo más recomendable para Android, es libre y además es soportado por Google (ha sido utilizado por los desarrolladores de Google para crear Android). Puedes utilizar cualquier versión de Eclipse a partir de la 3.3.1.

Para instalar Eclipse hay que seguir los siguientes pasos:

1. Accede a la página http://www.eclipse.org/downloads/ y descarga la última versión de "Eclipse IDE for Java Developers". Verás que se encuentra disponible para los sistemas operativos más utilizados, como Windows, Linux y Mac OS.

2. Este *software* no requiere una instalación específica, simplemente descomprimir los ficheros en la carpeta que prefieras. Si así lo deseas puedes crear un acceso directo en el escritorio o en el menú inicio del fichero *eclipse.exe*.

3. Al arrancar Eclipse comenzará preguntándonos que carpeta queremos utilizar como *workspace*. En esta carpeta serán almacenados los proyectos que crees en Eclipse. Es importante que conozcas su ubicación para poder hacer copias de seguridad de tus proyectos.

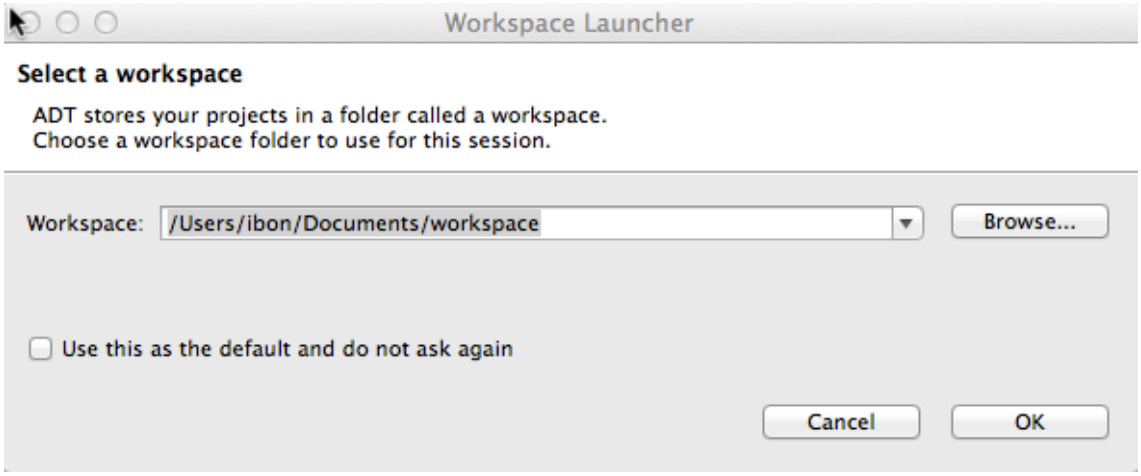


Figura 3.2: Seleccionar la carpeta workspace

3.2.3. Instalar Android SDK de Google

El siguiente paso consiste en instalar Android SDK de Google:

1. Accede a la siguiente página http://developer.android.com/sdk y descarga el fichero correspondiente a tu sistema operativo.

2. Este software no requiere una instalación específica, simplemente descomprimir los ficheros en la carpeta que prefieras.

3. Ejecuta el programa SDK Manager.

4. Seleccionar los paquetes a instalar. Aparecerá una ventana donde podremos seleccionar los paquetes a instalar. Si lo deseas puedes instalar todos los paquetes (Accept All), en este caso el proceso de instalación puede tardar más de una hora. Si no dispones de tanto tiempo puedes seleccionar solo algunos paquetes. Siempre resulta interesante instalar la última versión de Android (incluyendo documentación, ejemplos y por supuesto la plataforma). Más adelante podrás instalar más paquetes si necesitas otras plataformas de desarrollo u otras máquinas virtuales.

3.2.4. Instalación del plug-in Android para Eclipse (ADT)

El último paso consiste en instalar el plug-in Android para Eclipse, también conocido como ADT. Este software desarrollado por Google, instala una serie de complementos en Eclipse, de forma que el entorno de desarrollo se adapte al desarrollo de aplicaciones para Android. Se crearán nuevos botones, tipos de aplicación, vistas,... para integrar Eclipse con el Android SDK que acabamos de instalar.

1. Arranca Eclipse y selecciona Help>Install New Software…

2. En el diálogo Available Software que aparece, haz clic en Add… En el cuadro de diálogo Add Site que sale introduce la siguiente URL: http://dl-ssl.google.com/android/eclipse/

3. Selecciona los paquetes a instalar y pulsa Next. Ahora aparecen listadas las características de Android DDMS y Android Development Tools.

4. Pulsa Next para leer y aceptar la licencia e instalar cualquier dependencia y pulsa Finish.

5. Reinicia Eclipse.

6. Configura Eclipse para que sepa donde se ha instalado Android SDK. Para ello entra en las preferencias en Windows>Preferences… y selecciona Android del panel de la izquierda. Ahora pulsa Browse… para seleccionar el SDK Location y elige la ruta donde hayas descomprimido Android SDK. Aplica los cambios y pulsa OK.

3.2.5. Creación de un dispositivo virtual Android (AVD)

1. Abre Eclipse y pulsa en el botón Android Virtual Device Manager aparecerá la lista con los AVD que hayas creado. La primera vez estará vacía.

2. Pulsa a continuación el botón *New*... para crear un nuevo AVD. Aparecerá la siguiente ventana:

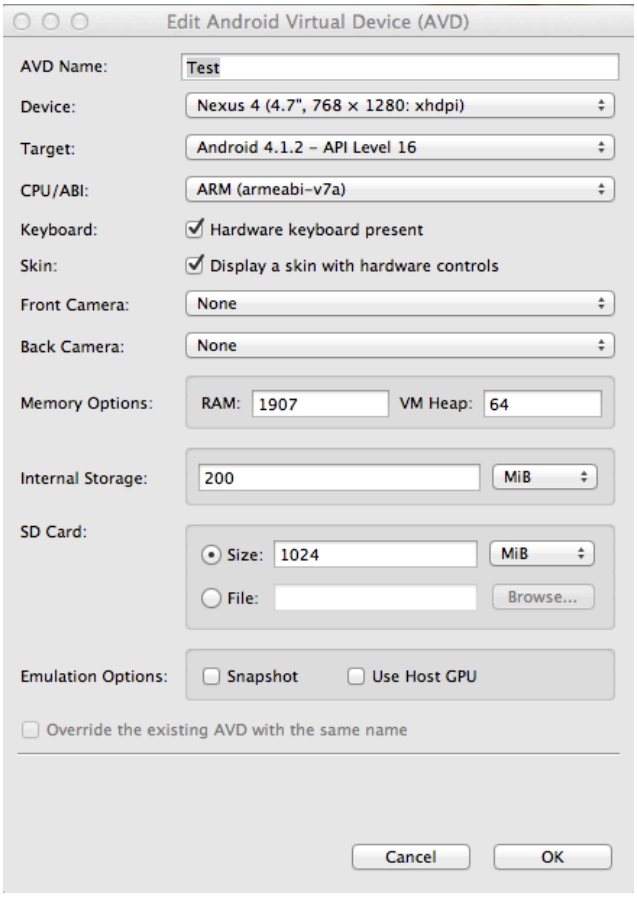


Figura 3.3: Creación de un nuevo AVD

Tendremos que introducir los siguientes datos:

- *AVD Name*: Nombre que quieras dar al nuevo dispositivo virtual.
- *Device*: Dispositivo a emular donde se indica el tamaño de la pantalla en pulgadas y la resolución del dispositivo
- *Target*: Versión SDK que soportará el dispositivo. Solo aparecerán las versiones que hayas instalado desde el Android SDK Manager.
- *CPU/ABI*: Tipo de CPU y arquitectura que se va a emular. La opción más habitual es ARM.
- *Keyboard*: Si se selecciona se supondrá que el dispositivo tiene teclado físico, que será emulado por el teclado del ordenador. En caso contrario se utilizará el teclado en pantalla.
- *Skin*: Si se selecciona se mostrarán a la derecha del dispositivo una serie de botones, entre los que se incluyen: volumen, on/off, teclas de navegación, retorno, home, menú…
- *Front/Back Camera*: Para activar la emulación de la cámara delantera y trasera.
- *Memory Options*: Memoria que se dedicará al emulador. RAM: memoria total en MB. VM Heap: Memoria dinámica asignada a la máquina virtual en MB.
- *Internal Storage*: Memoria interna del dispositivo. Determinará el número de aplicaciones y datos que podrás instalar.

- *SD Card*: Memoria externa del dispositivo. Size: tamaño de la memoria. Esta creará un nuevo fichero. File: Se utilizará un fichero previamente creado.
- *Snapshot*: Si lo seleccionas podrás congelar la ejecución del dispositivo en un determinado instante, sin tener que esperar a que se inicialice el dispositivo. Conviene marcarlo para conseguir una carga más rápida.
- *Use Host GPU*: Se habilita la emulación hardware para gráficos OpenGLES. Su navegación entre ventanas será más fluida.

3. Aparecerá el dispositivo creado en la siguiente lista. Para arrancarlo selecciónalo y pulsa el botón Start.

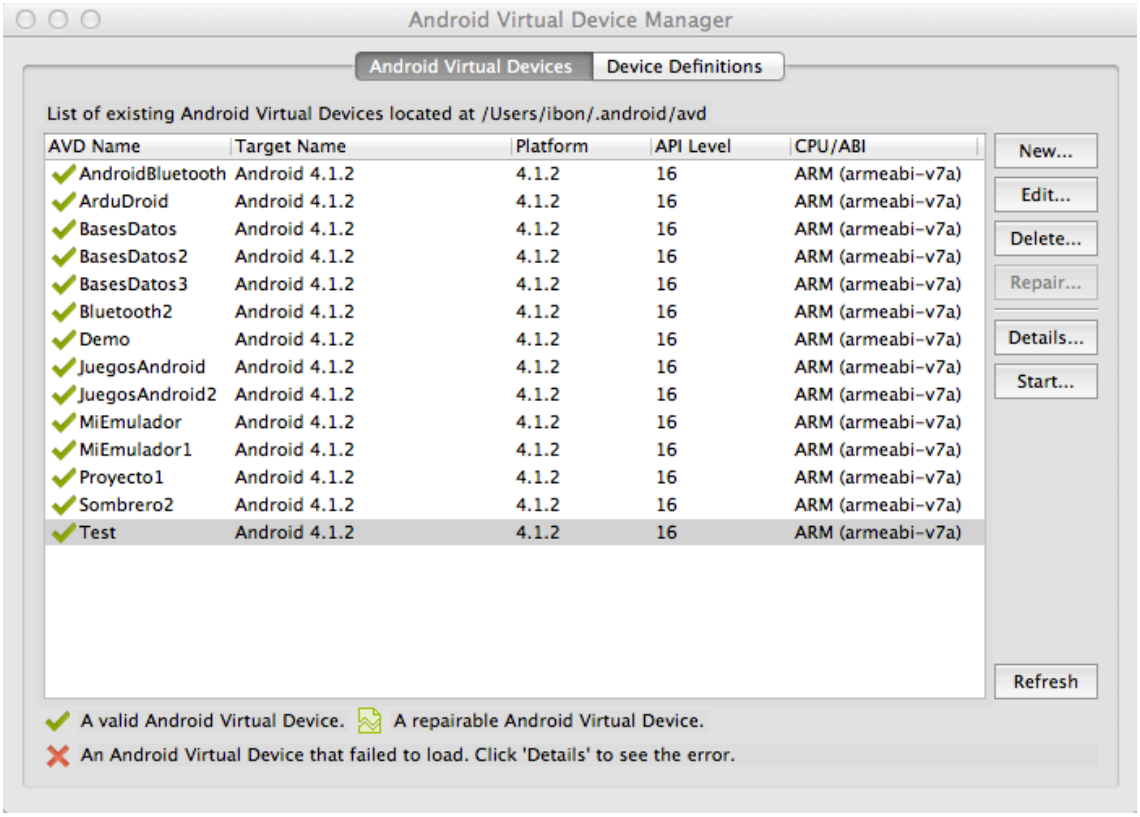


Figura 3.4: Lista de los dispositivos virtuales creados

Aparecerá la ventana Launch Options:

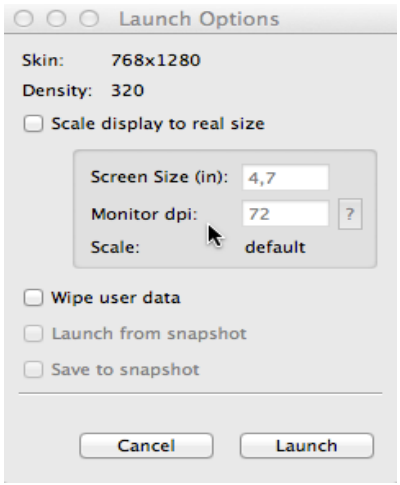


Figura 3.5: Launch Options

Puedes ejecutarlo en una ventana de 480x800 pixeles, o por el contrario, reescalarlo para que tenga un tamaño de 5,1 pulgadas en tu pantalla (*Scale display to real size*). También puede limpiar los datos de usuario (*Wipe user data*). Finalmente, puedes arrancar desde un punto de ejecución grabado e indicarle que cuando se cierre congele la ejecución para poder recargar en este mismo punto.

4. Pulsa el botón *Launch* para arrancarlo.

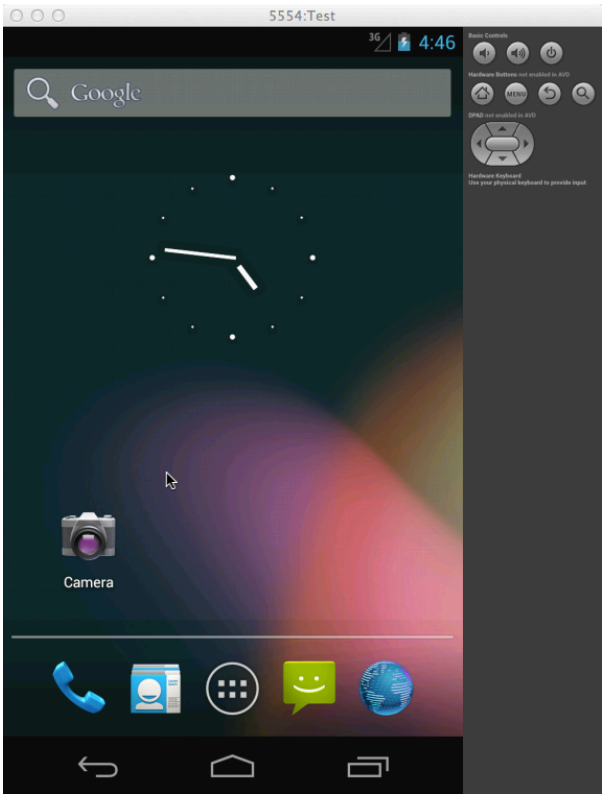


Figura 3.6: Emulador Android

4. Desarrollo e implementación

4.1. Creación de un proyecto en Eclipse

Una vez instalado Eclipse, el SDK y el ADT, ya tenemos todo lo necesario para poder desarrollar aplicaciones en Android. A continuación abrimos Eclipse y pulsaremos en File > New > Proyect. Dentro encontraremos una carpeta de nombre Android, dentro de esta seleccionaremos la opción Android Application Project. A continuación nos aparecerá la siguiente ventana:

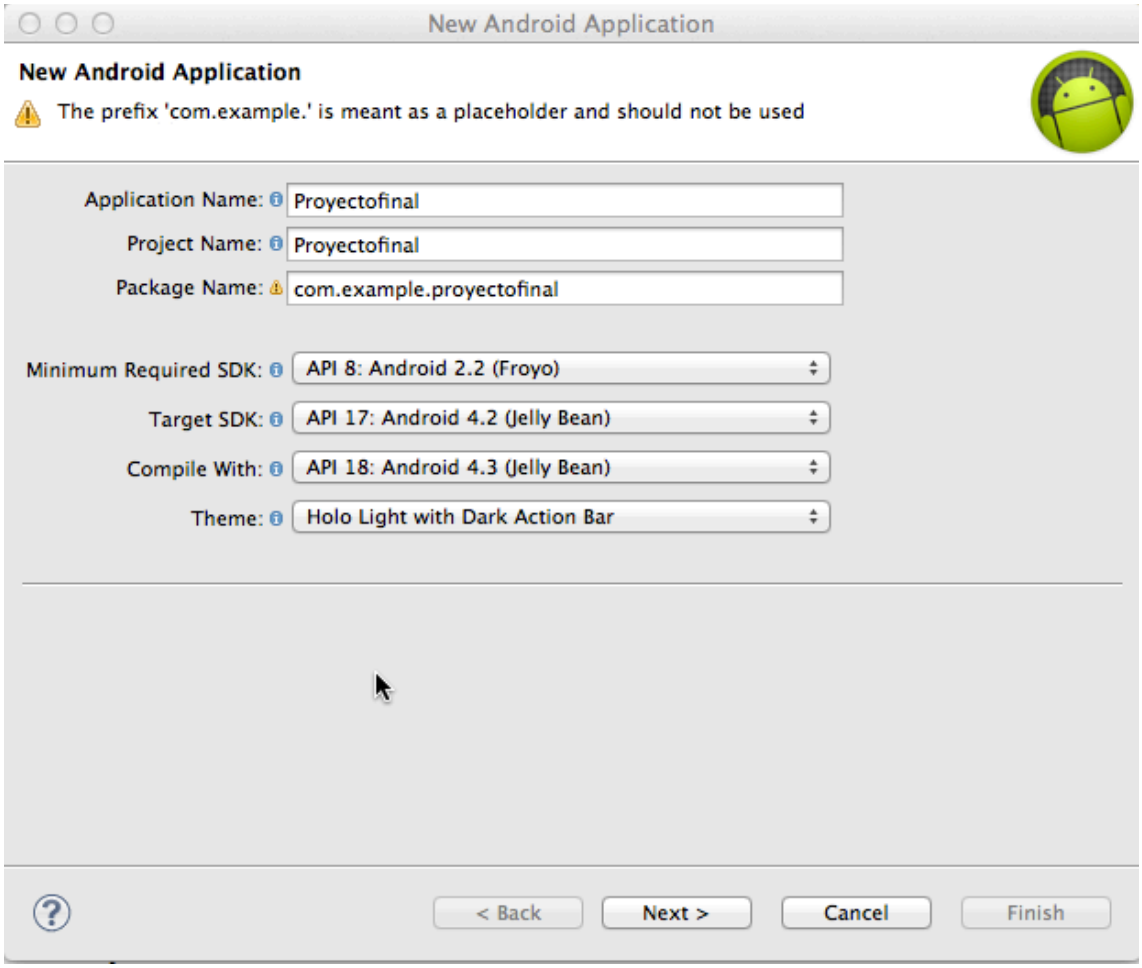


Figura 4.1: Creación de un nuevo proyecto Android I

A continuación se explican los campos que aparecen en la ventana emergente:

- *Aplication Name*: nombre de la aplicación.
- *Proyect Name*: nombre del proyecto Android.
- *Package Name*: nombre del paquete donde estará ubicado el proyecto.
- *Minimum Required SDK:* versión mínima de Android que soportara nuestra aplicación.
- *Target SDK*: versión máxima de Android que soportara nuestra aplicación.
- *Theme:* tema que tendrá por defecto nuestra aplicación.

Una vez que se han rellenado todos los campos, pulsamos *Next* y aparecerá un menú donde deberemos de indicar donde se creara el proyecto Android.

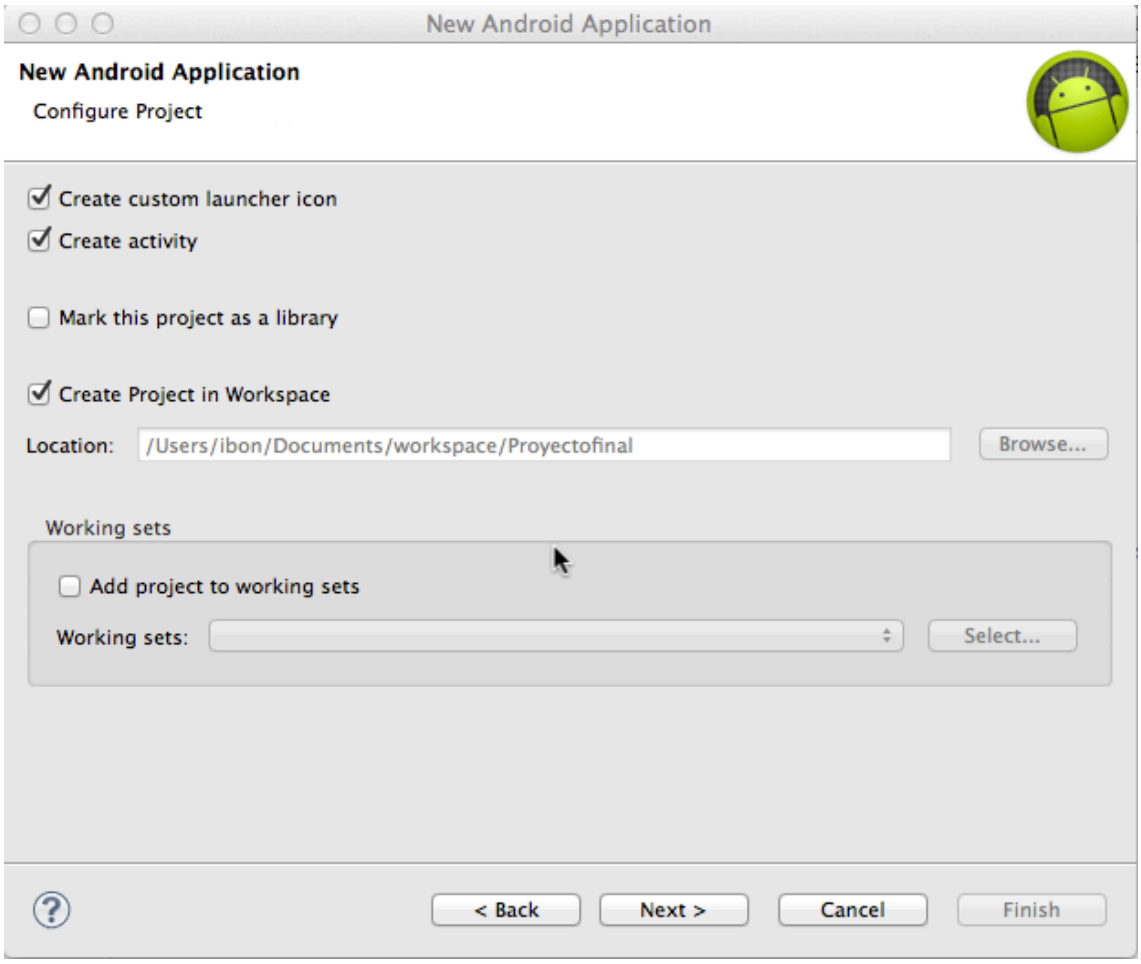


Figura 4.2: Creación de un nuevo proyecto Android II

En la siguiente pantalla del asistente configuraremos el icono que tendrá nuestra aplicación en el dispositivo. No nos detendremos mucho en este paso ya que no tiene demasiada relevancia por el momento. Tan sólo decir que podremos seleccionar la imagen, texto o dibujo predefinido que aparecerá en el icono, el margen, la forma y los colores aplicados. Por ahora podemos dejarlo todo por defecto y avanzar al siguiente paso pulsando *Next*.

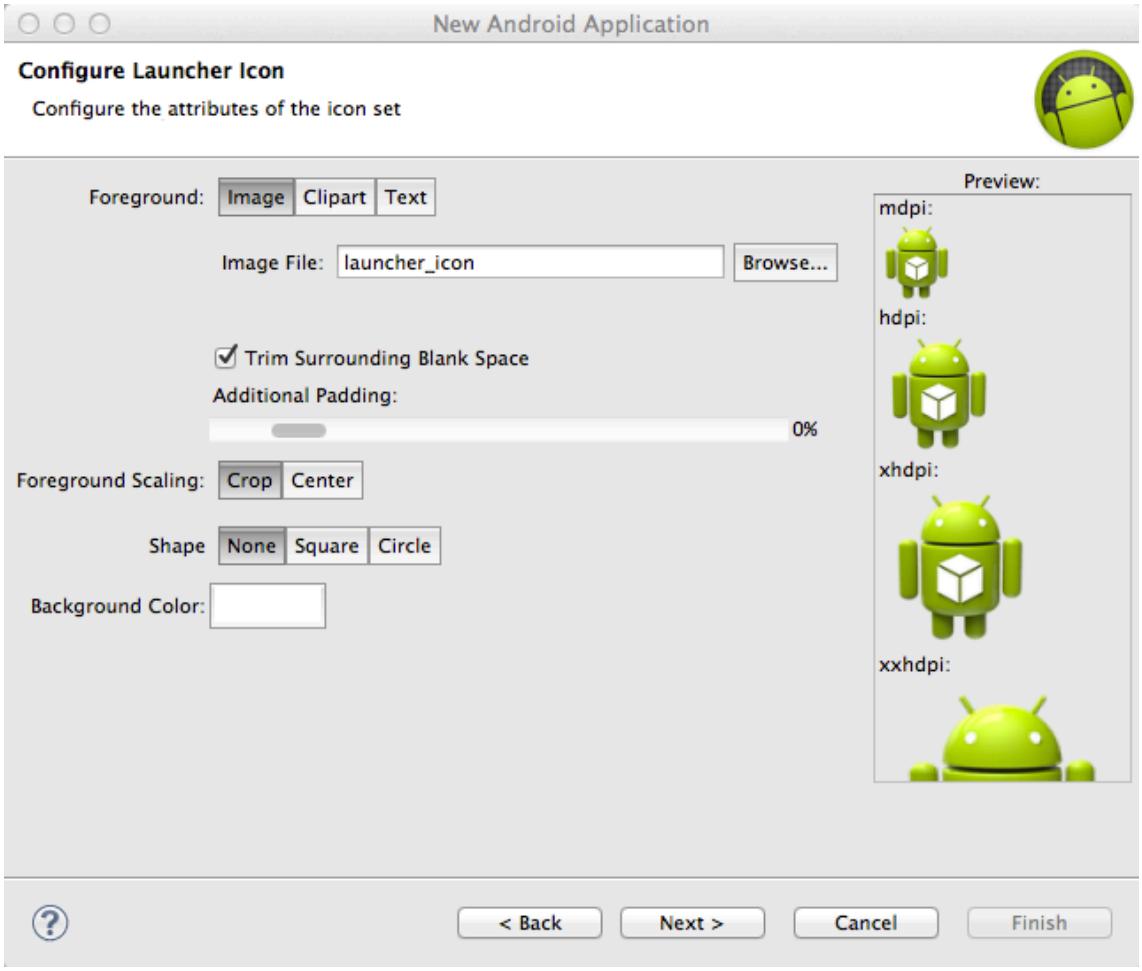


Figura 4.3: Creación de un nuevo proyecto Android III

En la siguiente pantalla del asistente elegiremos el tipo de Actividad principal de la aplicación. Entenderemos por ahora que una *actividad* es una "ventana" o "pantalla" de la aplicación. En este paso también dejaremos todos los valores por defecto, indicando así que nuestra pantalla principal será del tipo *BlankActivity*.

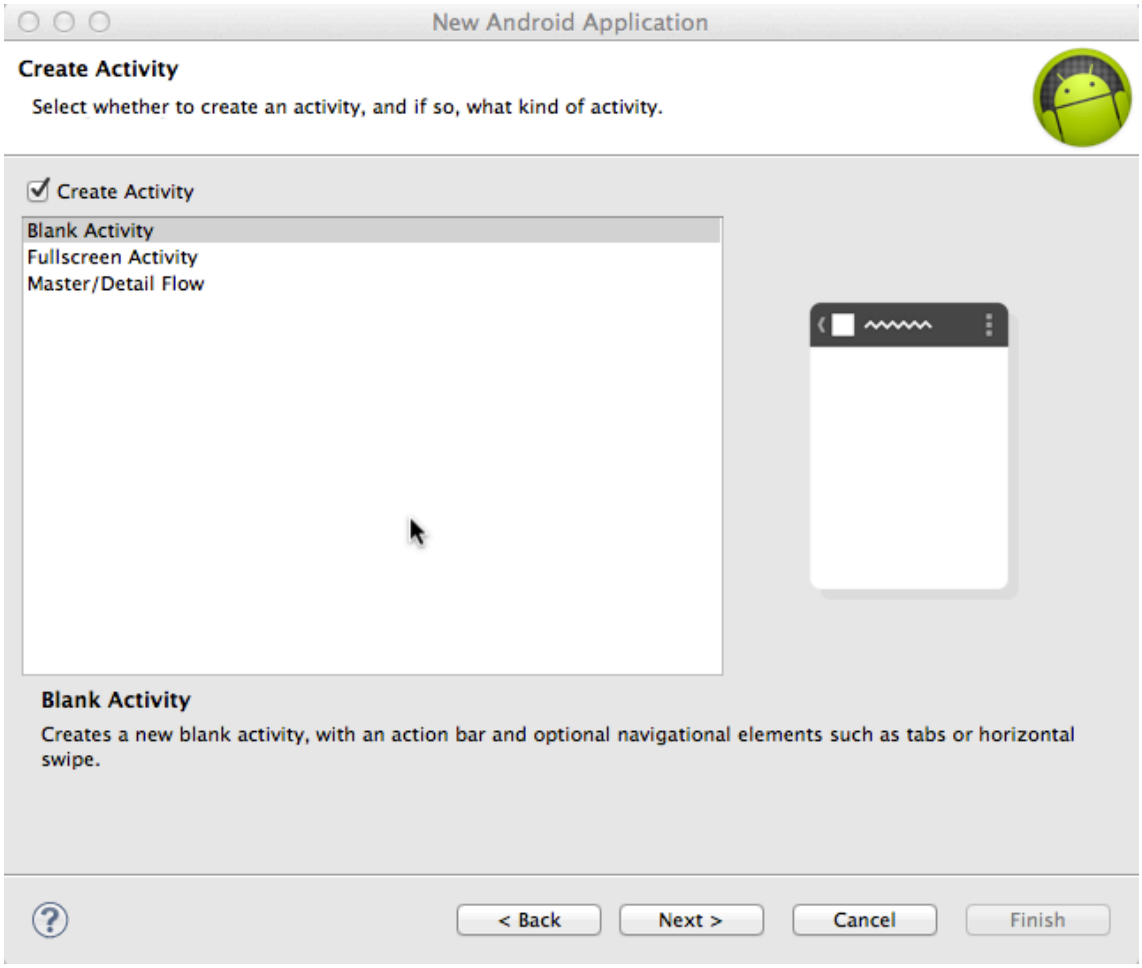


Figura 4.4: Creación de un nuevo proyecto Android IV

Por último, en el último paso del asistente indicaremos los datos de esta actividad principal que acabamos de elegir, indicando el nombre de su clase java asociada y el nombre de su *layout xml* (algo así como la interfaz gráfica de la actividad).

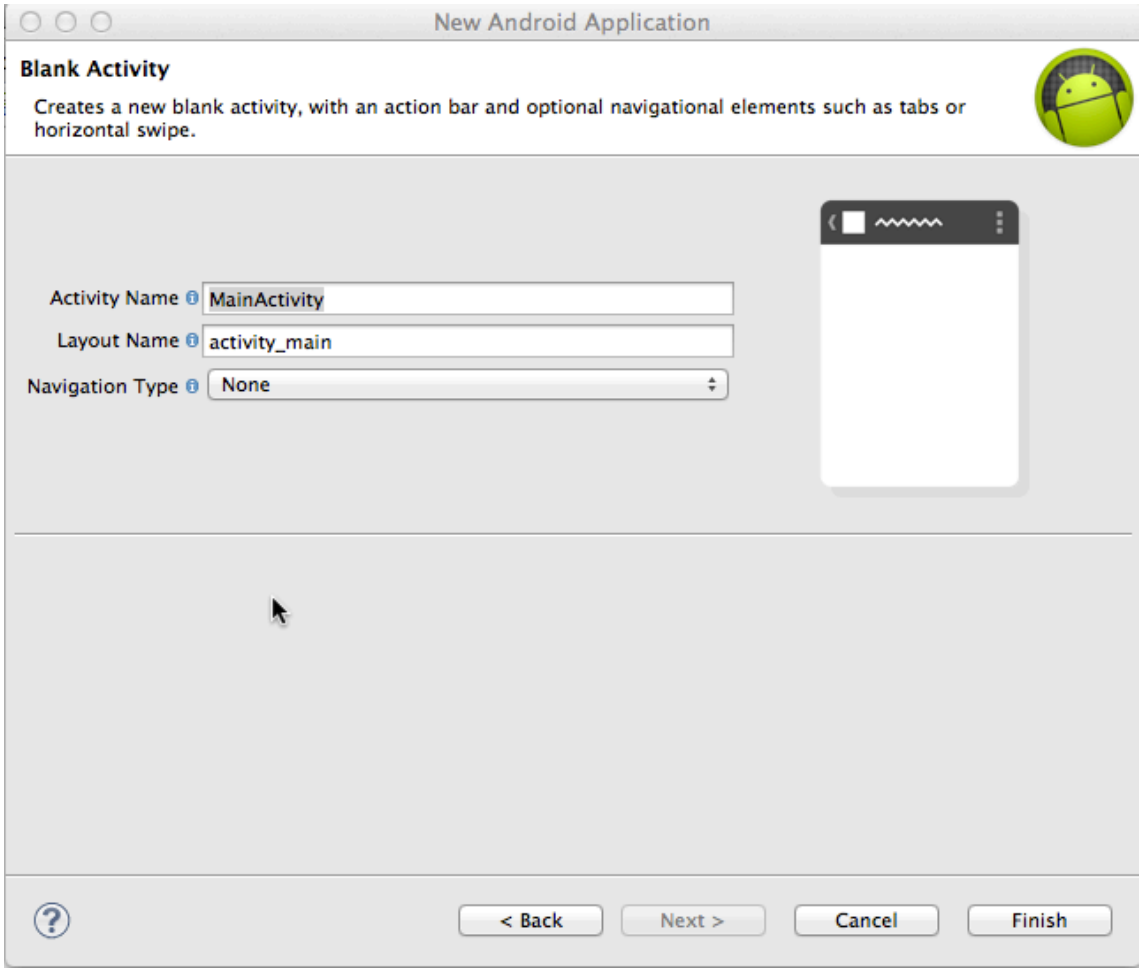


Figura 4.5: Creación de un nuevo proyecto Android V

Una vez configurado todo pulsamos el botón *Finish* y Eclipse creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener. En la siguiente imagen vemos los elementos creados inicialmente para un nuevo proyecto Android:

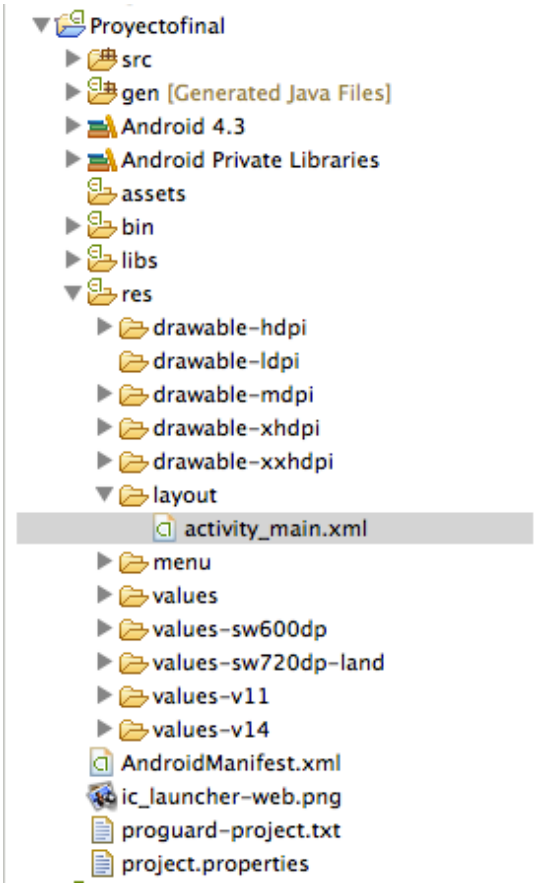


Figura 4.6: Estructura de carpetas de un proyecto Android

En los siguientes apartados describiremos los elementos principales de esta estructura.

- Carpeta *src*: Esta carpeta contendrá todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc. Inicialmente, Eclipse creará por nosotros el código básico de la pantalla (Activity) principal de la aplicación, que recordemos que en nuestro caso será MainActivity, y siempre bajo la estructura del paquete java definido. Esta carpeta también contendrá todos los paquetes de nuestra aplicación. En cada paquete estarán los diferentes archivos JAVA implementados.

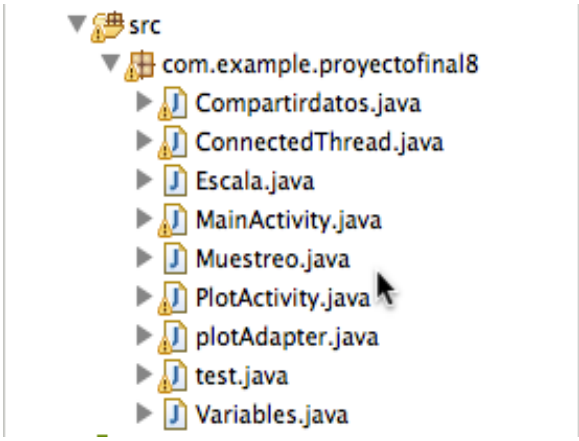


Figura 4.7: Estructura de carpeta src

- Carpeta *res*: Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se distribuyen entre las siguientes subcarpetas:
  Carpeta */res/drawable/*: Contiene las imágenes y otros elementos gráficos usados por la aplicación. Para definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas:
  /drawable-ldpi (densidad baja)
  /drawable-mdpi (densidad media)
  /drawable-hdpi (densidad alta)
  /drawable-xhdpi (densidad muy alta)
  Carpeta */res/layout/*: Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica.
  Carpeta */res/menu/*: Contiene la definición XML de los menús de la aplicación.
  Carpeta */res/values/*: Contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (*strings.xml*), estilos (*styles.xml*), colores (*colors.xml*), arrays de valores (*arrays.xml*), etc.

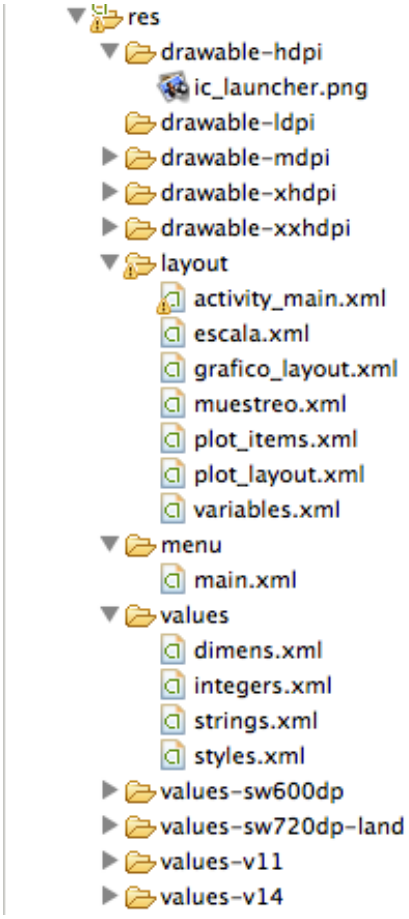


Figura 4.8: Estructura de carpeta res

Como se puede observar, existen algunas carpetas en cuyo nombre se incluye un sufijo adicional, como por ejemplo "values-v11" y "values-v14". Estos, y otros sufijos, se emplean para definir recursos independientes para determinados dispositivos según sus características. De esta forma, por ejemplo, los recursos incluidos en la carpeta "values-v11" se aplicarían tan sólo a dispositivos cuya versión de Android sea la 3.0 (API 11) o superior. Al igual que el sufijo "–v" existen otros muchos para referirse a otras características del terminal.

- Carpeta *gen*: Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente java dirigidos al control de los recursos de la aplicación. Importante: dado que estos ficheros se generan automáticamente tras cada compilación del proyecto es importante que no se modifiquen manualmente bajo ninguna circunstancia.

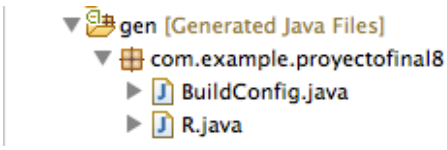


Figura 4.9: Estructura de carpeta gen

A destacar sobre todo el fichero que aparece desplegado en la imagen anterior, llamadoR.java, donde se define la clase R.
Esta clase R contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta /res/, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato. Así, por ejemplo, la constante R.drawable.ic_launcher contendrá el ID de la imagen "*ic_launcher.png*" contenida en la carpeta /res/drawable/.

- Carpeta *assets*: Contiene todos los demás ficheros auxiliares necesarios para la aplicación (y que se incluirán en su propio paquete), como por ejemplo ficheros de configuración, de datos, etc. La diferencia entre los recursos incluidos en la carpeta /res/raw/ y los incluidos en la carpeta /assets/ es que para los primeros se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Para los segundos sin embargo no se generarán ID y se podrá acceder a ellos por su ruta como a cualquier otro fichero del sistema. Usaremos uno u otro según las necesidades de nuestra aplicación.

- Carpeta *bin*: Ésta es otra de ésas carpetas que en principio no tendremos por qué tocar. Contiene los elementos compilados de la aplicación y otros ficheros auxiliares. Cabe destacar el fichero con extensión ".apk", que es el ejecutable de la aplicación que se instalará en el dispositivo.

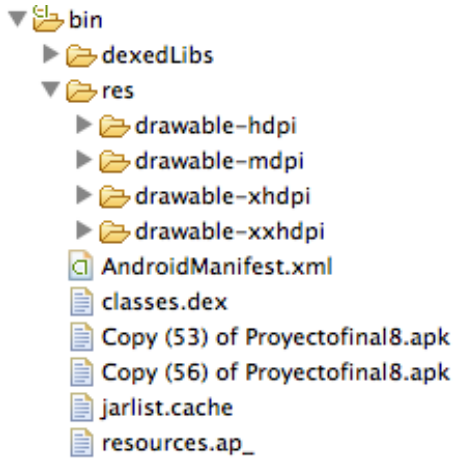


Figura 4.10: Estructura de carpeta bin

- Carpeta *libs*: Contendrá las librerías auxiliares, normalmente en formato ".jar" que utilicemos en nuestra aplicación Android.

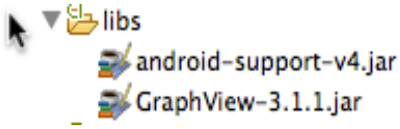


Figura 4.11: Estructura de carpeta libs

- Fichero *AndroidManifest.xml*: Contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono, …), sus componentes (pantallas, mensajes, …), las librerías auxiliares utilizadas, o los permisos necesarios para su ejecución. Veremos más adelante más detalles de este fichero.

4.2. Archivos

A lo largo del desarrollo de una aplicación Android, nos encontraremos con numerosos archivos xml (interfaz gráfica) y archivos java (funcionalidad de la aplicación).

4.2.1. Archivos XML

Para desarrollar la interfaz gráfica de una actividad se puede hacer de manera más intuitiva y sencilla mediante archivos XML. Cada archivo XML está compuesto por un árbol de elementos que especifican la manera en la que los elementos de la UI y contenedores se acomodaran para definir la parte visual de un objeto *View*. Para la creación de estos archivo Eclipse ofrece dos posibilidades, una de forma visual (Graphical Layout) y otra en modo texto (archivo.xml).

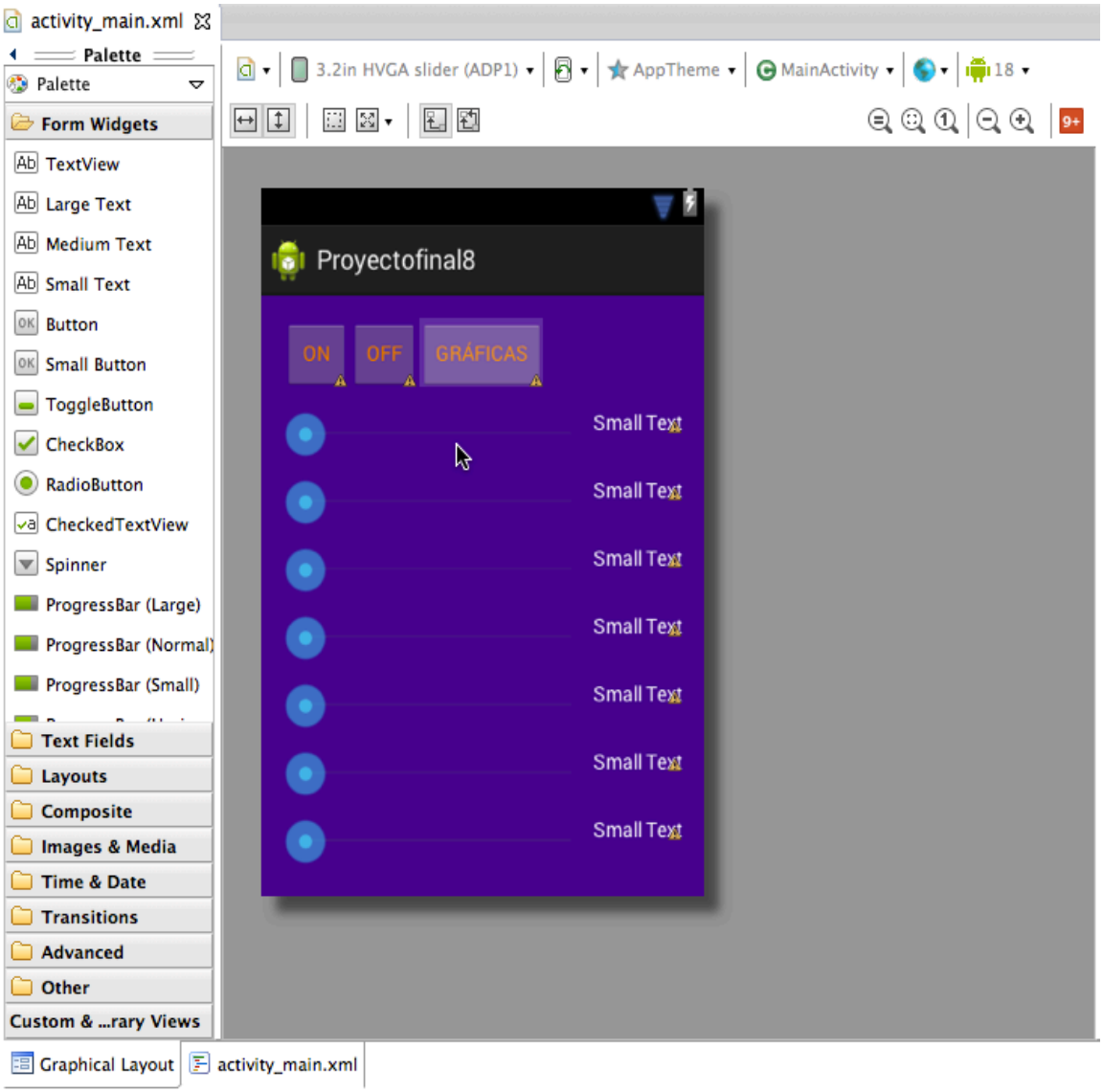


Figura 4.12: Fichero XML (Graphical Layout)

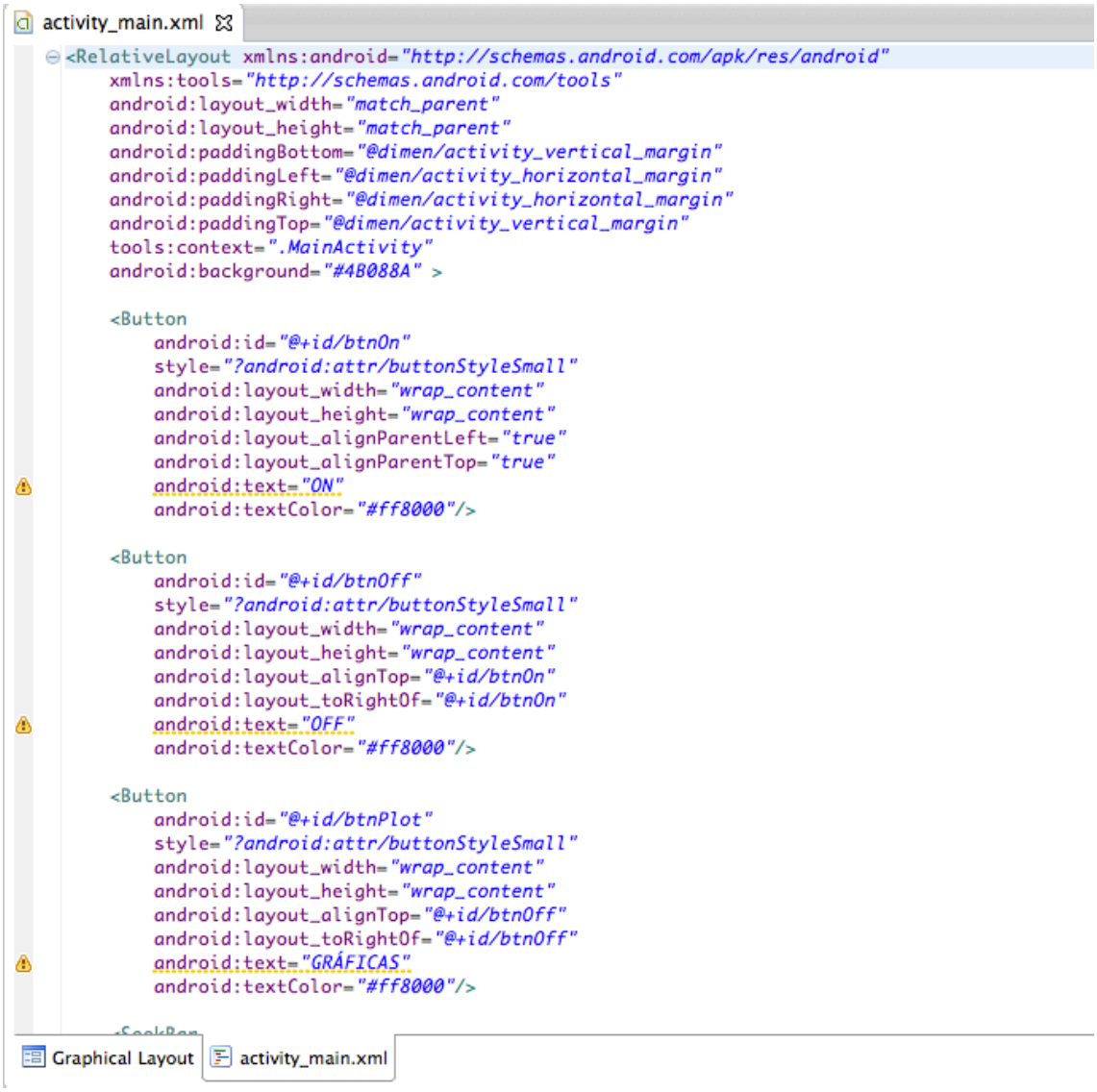


Figura 4.13: Fichero XML (activity_main.xml)

A continuación vamos a analizar el contenido de los archivos XML de la aplicación:

- ***activity_main.xml:*** Este archivo XML está compuesto por 3 botones (Button), 7 barras progresivas (SeekBar) y 7 Textos pequeños (Small Text) todo ello en forma de RelativeLayout de forma que están colocados aleatoriamente y no linealmente uno debajo del otro.

- ***grafico_layout.xml:*** Este archivo XML no contiene nada, lo que realmente hace que cobre sentido es la librería instalada en la carpeta *libs* (*GraphView-3.1.1.jar)* y la clase test.java que llama a esta librería con la finalidad de crear una vista en la que se genera una gráfica con los ejes horizontal y vertical.

- ***plot_items.xml:*** En este archivo XML se representa el espacio el cual va a ser ocupado por la lista de elementos que reflejaran las distintas variables de los sensores para acceder a su gráfica correspondiente y el icono que aparecerá en

cada uno de estos elementos o ítems que en este caso será el mismo que se eligió a la hora de crear el proyecto.

- ***plot_layout.xml:*** Este archivo XML contiene la lista de variables a las que podemos acceder para visualizar su correspondiente gráfica en tiempo real.

- ***muestreo.xml:*** Este archivo XML contiene un textView, un editText para introducir el tiempo de muestreo y un botón para enviar el dato.

- ***escala.xml:*** Este archivo XML no contiene ningún elemento. Quedará para futuras líneas de investigación.

- ***variables.xml: :*** Este archivo XML no contiene ningún elemento. Quedará para futuras líneas de investigación.

4.2.1.1. AndroidManifest

Se trata de un archivo XML que estará en todas las aplicaciones Android. En él se declaran todas las especificaciones de nuestra aplicación. *Activities, Service, Intents,* bibliotecas, el nombre de la aplicación, el hardware que se necesita, los permisos de la aplicación…

Vamos a definir los principales tags del manifiesto:

- <manifest>: tag raíz. En él se declara el número de versión de desarrollo, el número de versión de nuestra aplicación y el paquete raíz que lo contiene.

- <application>: tag que contiene todas las Activities, Services, Providers, Receivers y las bibliotecas que se usan en nuestra aplicación.

- <uses-permissions>: mediante este tag especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse, además son los que deberá aceptar el usuario antes de instalarla. Por ejemplo, si se desea utilizar funcionalidades como en este caso, bluetooth, hay que indicar que nuestra aplicación requiere esos permisos.
  En mi caso utilizaré los siguientes permisos bluetooth que son imprescindibles para el correcto funcionamiento de la aplicación:

```
<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission
android:name="android.permission.BLUETOOTH" />
```

- <uses –sdk>: tag utilizado para determinar las distintas versiones Android que va a utilizar nuestra aplicación, tanto sobre qué versiones va a correr como qué versión fue utilizada para realizar nuestras pruebas. Mediante el atributo *android:minSdkVersion* establecemos a partir de qué versión de Android podrá correr nuestra aplicación.

- <activity>: es imprescindible añadir todas y cada una de las activities que van a ser utilizadas en la aplicación para que el manifest tenga conciencia de que esas activities existen y van a ser implementadas. El orden de colocación de las activities en el AndroidManifest.xml influirá en que clase se va a implementar cuando se ejecute la aplicación.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.proyectofinal8"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:name="com.example.proyectofinal8.Compartirdatos"
        >
        <activity
            android:name=".MainActivity"
            android:label="proyectofinal8"
            android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            android:screenOrientation="portrait"
            >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
         <activity
            android:name=".PlotActivity"
            android:label="proyectofinal8"
            android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            >


        </activity>
         <activity
            android:name=".test"
            android:label="proyectofinal8"
```

Figura 4.14: Fichero AndroidManifest.xml

4.2.2. Archivos JAVA

Funcionamiento e implementación de clases. En este apartado se verán las diferentes clases que componen la aplicación:

- ***Compartirdatos.java:*** Voy a empezar explicando esta clase antes que la clase principal MainActivity.java ya que es la encargada de compartir los datos que van a ser indispensables para el resto de Activities.
  Esta clase surgió porque al pasar de una clase a otra, la conexión se perdía ya que solo la clase MainActivity.java acogía la conexión bluetooth entre

dispositivos y al entrar en estado onPause() todo el contenido de la clase dejaba de estar activo.
Otro problema es que los datos de los sensores de Arduino los necesitábamos en diferentes Activities para implementar la gráfica de las variables a tiempo real.
Esta clase nos permite compartir tanto la conexión bluetooth, como los distintos datos de las variables de tal forma que cualquier otra clase pueda acceder a ellos.
De esta forma *Compartirdatos.java* comparte datos con las demás clases para que en ningún momento se pierda la conexión bluetooth y cualquier clase pueda acceder en cualquier instante a los datos que se reciben de los sensores.

- ***ConnectedThread.java:*** Extiende de la clase *Thread.* Un *Thread* es una unidad de ejecución concurrente. Tiene su propia pila de llamadas de métodos que se invocan, sus argumentos y variables locales. Esta aplicación tiene un subproceso en ejecución cuando se inicia el hilo principal.

- ***MainActivity.java:*** Se trata de la actividad principal. Relacionada con el archivo *activity_main.xml*. Esta clase es la encargada de recoger los datos que aporta Arduino mediante Bluetooth una vez ha llamado a la clase *Compartirdatos.java.*
  Los botones btnOn y btnOff realmente no tienen ninguna funcionalidad en nuestro programa, se colocaron para poder tener control sobre el envío de datos que quedará para líneas de investigación futuras.
  El botón btnPlot esta colocado para que una vez pulsado acceda a una lista de elementos en la que se podrá elegir a qué gráfica de qué variable acceder.
  Las barras progresivas representan gráficamente los valores que reciben los sensores desde arduino, sus valores oscilan entre 0 y 1023.
  Cada valor de cada variable se colocará en su SeekBar correspondiente.
  El valor 1 en la seek1, el valor 2 en la seek 2 y así sucesivamente.
  Los textView representan numéricamente los valores que reciben los sensores desde arduino.

  Para que asimile el programa una trama de datos como correcta, han de llegar precedidos por el símbolo "<" y finalizar con el símbolo ">", de tal forma que un string de datos sería por ejemplo de la siguiente forma: <193123212410001822331 35>
  El primer byte después del símbolo "<" representa la longitud que ocupan los datos de la variable 1, en este caso el valor de los datos de la variable 1 será 9.
  El siguiente byte que indicará la longitud de los datos de la variable 2 será el valor 3, por lo tanto el valor de los datos de la variable 2 será el 123 y así sucesivamente hasta encontrarnos con el símbolo ">".
  La trama siempre estará compuesta por el símbolo "<", 7 longitudes de datos con sus correspondientes valores y el símbolo ">" como fin de la trama recibida.
  También se genera un buffer en el que se van guardando hasta 20 valores de cada variable recibida que se van reciclando para su posterior representación en la gráfica a tiempo real.

  En esta clase también se crea un menú con las opciones *Muestreo, Escala y Variables* que se implementó para el envío de comandos a Arduino pero que no tienen ninguna aplicación real y que quedará para futuras líneas de investigación.

- ***test.java:*** Esta actividad es la encargada de dibujar las gráficas correspondientes de los valores de cada sensor en tiempo real. Los valores se recogen de la Activity Compartidos.java, test.java llama en varias ocasiones a Compartidos. java cada vez que necesita recoger algún dato que tenemos almacenado en esta Activity. El eje vertical representa los valores recogidos de los sensores. Estos valores estarán siempre comprendidos entre 0 y 1023. Este eje varía su escala en función de los datos recibidos para tener una visión más clara de los cambios bruscos producidos en los valores de los sensores. El eje horizontal representa el tiempo en el que se recogen los valores de los sensores, una muestra por segundo. El corazón de esta Acivity es la librería que hemos incorporado en la carpeta *libs* (*GraphView-3.1.1.jar)*. Gracias a esta librería la Acticity cobra sentido. Sin ella el funcionamiento de esta clase no sería posible. La idea de incorporar librerías en nuestra aplicación facilita mucho el trabajo ya que muchas de las funciones que se desean incorporar en nuestra aplicación puede que ya las hayan creado anteriormente.

- ***PlotActivity.java:*** Se trata de la actividad que da sentido a la lista de elementos de las sensores utilizados en nuestra aplicación. Esta clase da nombre a esta lista de elementos. Cuando seleccionas el elemento de la lista de la variable deseada te lleva a la Activity test.java en la que se verán representados los valores correspondientes a ese sensor como acabamos de explicar. Está clase está relacionada con el archivo *plot_layout.xml.*

- ***plotAdapter.java:*** Extiende de la clase BaseAdapter. La clase BaseAdapter de una aplicación es común para un Adapter que se puede utilizar tanto en *ListView* (mediante la implementación de la interfaz especializada ListAdapter) como en un *Spinner* (por la aplicación de la interfaz especializada SpinnerAdapter).
  Proviene de la clase *java.lang.Object* y de las interfaces *android.widget.Adapter, android.widget.ListAdapter* y *android.widget.SpinnerAdapter.*

- ***Muestreo.java:*** Esta actividad aunque fue creada para darle una funcionalidad, en realidad no tiene ninguna. Estaba pensada para el envío de datos desde un dispositivo móvil a Arduino con la finalidad de modificar el tiempo de refresco para la lectura de las variables y quedará para futuras líneas de investigación.

- ***Escala.java:*** Esta actividad aunque fue creada para darle una funcionalidad, en realidad no tiene ninguna. Estaba pensada para el envío de datos desde un dispositivo móvil a Arduino con la finalidad de modificar el escalado de la gráfica y quedará para futuras líneas de investigación.

- ***Variables.java:*** Esta actividad al igual que las anteriores fue creada para darle una funcionalidad pero en realidad no tiene ninguna. Estaba pensada para el envío de datos desde un dispositivo móvil a Arduino con la finalidad de modificar las variables muestreadas y quedará para futuras líneas de investigación.

5. Instalación y uso

5.1. Instalación de la aplicación

Se han utilizado dos opciones para instalar la aplicación:

- La primera opción utilizada fue la instalación desde el archivo APK: Para crear el archivo ejecutable *apk* de un proyecto Android se puede hacer seleccionando la opción de Eclipse *Import* o simplemente ejecutando el proyecto en un dispositivo virtual Android (*AVD Manager)*. Una vez generado el archivo lo guardaremos en cualquier carpeta del dispositivo móvil. Hay que ejecutar el archivo para poder instalarlo. Será necesario tener activado en el dispositivo móvil la opción *orígenes desconocidos*, que nos permitirá instalar aplicaciones que no hayan pasado por la tienda virtual de Android (Android Market).

- Posteriormente, se llegó a la conclusión de que la forma más funcional para instalar la aplicación era realizarla desde Eclipse: Esta es la opción más rápida y más eficaz no solo para instalar la aplicación una vez este finalizada, sino para ir realizando las pruebas necesarias y detectar rápidamente cualquier error mientras se está trabajando con Eclipse. Es necesario conectar el dispositivo móvil al PC mediante su cable USB. Al igual que si se tratara de un dispositivo virtual, nos aparecerá en el AVD Manager. Hay que pinchar en la opción de Eclipse *Run as* y seleccionar nuestro dispositivo. También hay que tener activado en el dispositivo móvil la opción *orígenes desconocidos*.

5.2. Uso de la aplicación

Una vez seleccionada nuestra aplicación y comience a funcionar el programa, primero se establecerá la conexión inalámbrica bluetooth entre el dispositivo móvil y Arduino gracias al módulo bluetooth LM780. Vemos que esta conexión se produce porque el led incorporado en la placa de montaje conectado a Arduino deja de parpadear y mantiene una iluminación constante. Si el led vuelve a parpadear quiere decir que se ha perdido la conexión bluetooth entre el dispositivo móvil y Arduino.

Una vez abierta la aplicación y establecida la conexión inalámbrica bluetooth, esta empieza a recibir los datos de los sensores cuyos valores oscilan entre 0 y 1023. Los valores se reflejan tanto en las siete barras progresivas correspondientes a cada una de las variables como numéricamente a la derecha de cada barra progresiva.

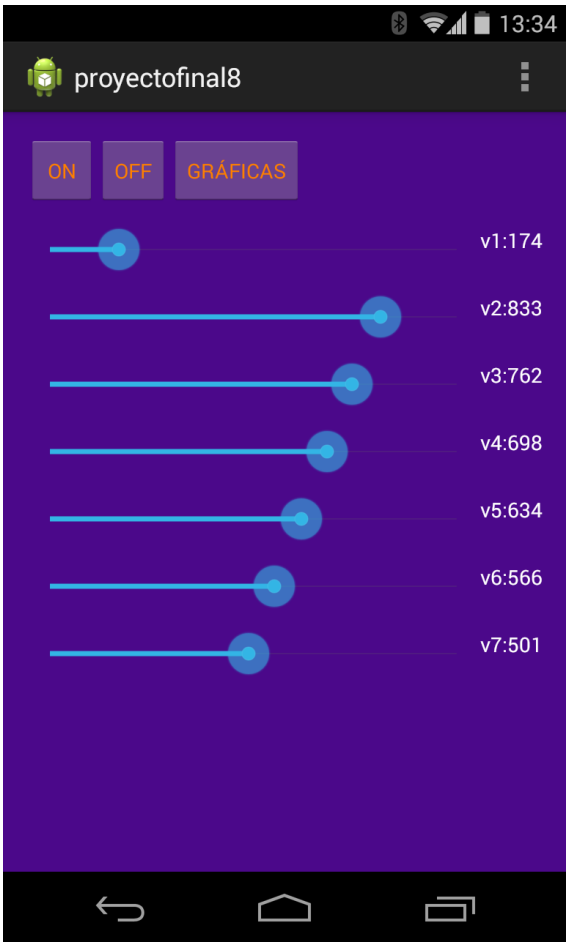


Figura 5.1: Imagen de la Activity principal

Si pulsamos el botón *GRÁFICAS* accedemos a otra Activity. Recordamos que la conexión bluetooth y los datos recibidos de la Activity principal siguen en funcionamiento ya que el archivo *Compartirdatos.java* comparte datos con las demás clases para que en ningún momento se pierda la conexión y cualquier clase pueda acceder en cualquier instante a los datos que se reciben de los sensores. La Activity a la que accedemos está compuesta por una lista de elementos en la cual cada elemento da la posibilidad de acceder a la gráfica en tiempo real de su variable correspondiente.

Figura 5.2: Imagen de la Activity con la lista de elementos

Una vez decidamos la variable a la cual queremos acceder a su gráfica en tiempo real, seleccionamos el elemento correspondiente a dicha variable y pasaremos a la siguiente Activity.

La siguiente Activity muestra la gráfica en tiempo real de la variable correspondiente. El eje vertical representa los valores recibidos de cada sensor entre 0 y 1023. Se trata de un eje en el que la escala se ajusta en función de los datos recibidos, es decir, si solo aparecen valores entre 328 y 348, en el eje vertical se reflejara como valor mínimo el 328 y como valor máximo el 348, si en la siguiente muestra se recoge el valor 900, el eje vertical variara siendo ahora el valor mínimo 328 y el valor máximo 900. El eje horizontal representa el tiempo en el que se recogen los valores de cada muestra de los sensores, en este caso se recoge una muestra por segundo. Las muestras recogidas en las gráficas son progresivas. Se recogen hasta 10 muestras, a partir de aquí cuando entra la muestra onceava, se descarta la primera muestra y así sucesivamente.

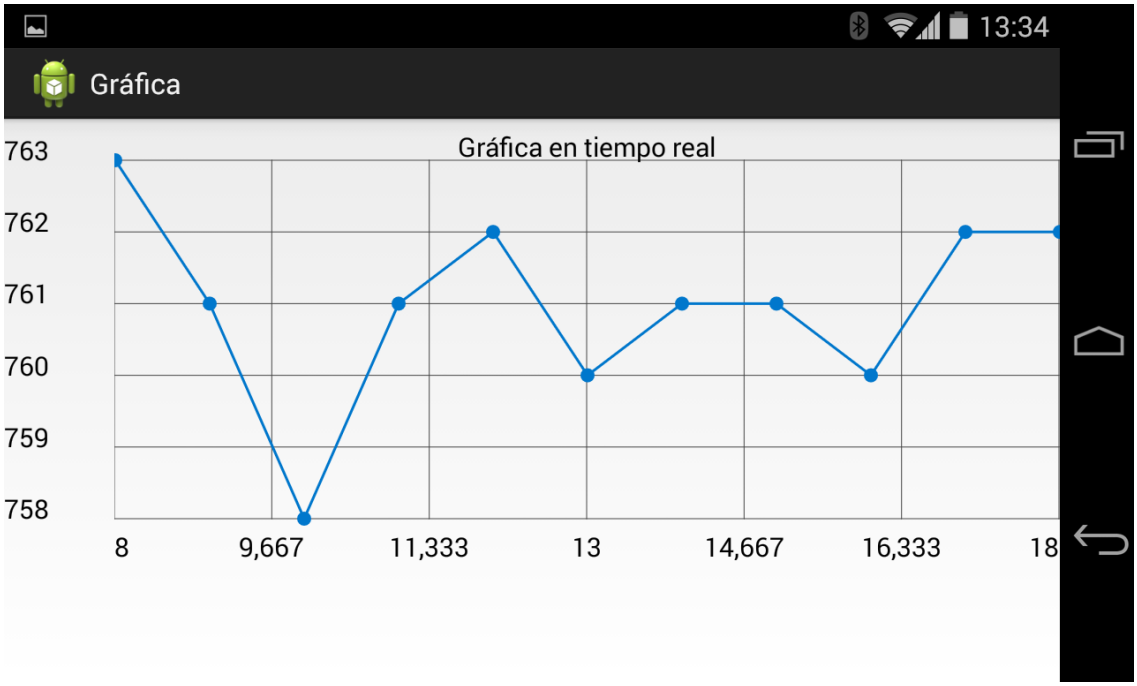


Figura 5.3: Imagen de la Activity que representa la gráfica

Figura 5.4: Imagen del desarrollo del proyecto

6. Desarrollo del proyecto

La realización del proyecto se desarrollo en las siguientes fases:

- **Fase 1:** Familiarizarse con el entorno de programación Eclipse. Realización de pequeños programas para entender el dinamismo de la plataforma Android.

- **Fase 2:** Desarrollo y montaje de la placa.

- **Fase 3:** Análisis de requerimientos y diseño de las interfaces gráficas y las funciones que realizará la aplicación.

- **Fase 4:** Realizar la conexión inalámbrica bluetooth mediante una aplicación Android entre el dispositivo móvil y arduino. (Enviar y recibir datos).

- **Fase 5:** Desarrollo de la Activity principal en la que se reciben los datos en las barras progresivas.

- **Fase 6:** Desarrollo de la Activity de la lista de elementos y de la representación gráfica de los datos recibidos en tiempo real.

- **Fase 7:** Desarrollo de la memoria y preparación de la presentación (powerpoint).

TAREA	Duración determinada
Familiarizarse con el entorno de programación Eclipse	35 días
Desarrollo y montaje de la placa	5 días
Plantear interfaces gráficas y funciones de la aplicación	5 días
Realizar la conexión inalámbrica bluetooth entre el dispositivo móvil y Arduino	30 días
Desarrollo de la actividad principal	25 días
Desarrollo de las actividades secundarias	20 días
Desarrollo de la memoria y preparación de la presentación	25 días
TOTAL	145 días

Figura 6.1: Duración de las tareas realizadas

7. Presupuesto del proyecto

En este apartado se exponen las tablas de los gastos originados durante el transcurso del proyecto, tanto el gasto de los materiales como el gasto de la mano de obra. En todos los precios está incluído el 21% de IVA. La fase 1, familiarización con el entorno de programación Eclipse se ha excluido de la mano de obra ya que es trabajo previo que no se le cobrará al cliente.

Materiales	Unidades	Precio unidad	€
Arduino MEGA 2560	1	48	48
Módulo bluetooth LM780	1	23	23
Resistencias	12	0.2	2.4
Diodos LED	3	0.7	2.1
Pulsador	1	0.45	0.45
Condensadores	2	0.3	0.6
Conectores de paso	1	0.5	0.5
Placa de impresión PCB	1	6	6
Cinta termo retráctil	1	8	8
TOTAL			91.05

Figura 7.1: Precio de los materiales utilizados

Mano de obra	Horas	Precio / hora	€
Desarrollo y montaje de la placa	10	15	150
Plantear interfaces gráficas y funciones de la aplicación	10	15	150
Realizar la conexión inalámbrica bluetooth entre el dispositivo móvil y Arduino	60	15	900
Desarrollo de la actividad principal	50	15	750
Desarrollo de las actividades secundarias	40	15	600
Desarrollo de la memoria y preparación de la presentación	50	15	750
TOTAL	290	15	3300

Figura 7.2: Precio de la mano de obra empleada

Materiales	91.05 €
Mano de obra	3.300 €
TOTAL	3391.05

Figura 7.3: Precio total

8. Conclusiones y líneas futuras

8.1. Conclusiones

La idea original del proyecto era realizar la conexión inalámbrica bluetooth entre un dispositivo móvil y el PC simulando las variables aleatoriamente desde LabView. El proyecto original que mi compañero realizaba en paralelo sufrió variaciones, en las que se decidió realizar la comunicación inalámbrica bluetooth directamente entre Arduino y LabView. Por tanto, dados los requerimientos de diseño, eficiencia y funcionalidad, se decidió que era más conveniente realizar la conexión inalámbrica bluetooth directamente entre un dispositivo móvil y Arduino. De esta manera los valores de las variables provienen de un dispositivo real, sin necesidad de simularlas aleatoriamente. Esta variación ha supuesto un poco más de trabajo, pero el resultado final ha merecido la pena.
En la siguiente lísta se detallan los principales logros de este proyecto:

- Se han alcanzado la mayoría de objetivos propuestos inicialmente, que eran el envío de comandos desde Arduino al dispostivo móvil. En la aplicación se reciben 7 señales DC y se pueden monitorizar como valores númericos. También se ha conseguido representar los valores recibidos de cada sensor en una gráfica a tiempo real en la que se recoge una muestra por segundo.

- El envío de comandos desde el dispositivo móvil a Arduino no ha sido igual, no se han conseguido alcanzar los objetivos propuestos inicialmente sobre todo por falta de tiempo y de recursos. Algunas de las opciones serían la implementación de los botones *ON* y *OFF* para habilitar o deshabilitar la conexión bluetooth, o implementar la funcionalidad del menú con las opciones *Muestreo, Escala* y *Variables* para modificar el tiempo de refresco de la lectura de variables, el escalado de las gáficas y la modificación de las distintas variables muestreadas. Todo ello quedará para futuras líneas de investigación.

- Como conclusión final mencionar que la parte más dura del proyecto fue la falta de recursos y de información a la que podíamos acceder. Estamos acostumbrados a buscar toda la información en internet y en Cuba carecíamos de él, lo cual hizo que cualquier duda por pequeña que fuera se hacía un mundo y tardaba bastante en resolverla. En lo referente a este proyecto, nadie en la universidad sabía como trabajar con la plataforma Android así que era muy difícil avanzar y solucionar problemas. Por el contrario, las dudas que surgían sobre la programación en java si que pudieron ser atendidas. La única información recibida eran los pdf's que adjuntaban tanto el tutor de este proyecto como amigos y antiguos alumnos a los que pedía información para que buscaran en internet sobre alguna duda concreta. Aún así avanzaba muy lento. Por fin conocí a una persona ajena a la universidad que conocía algo sobre la plataforma Android y me pudo resolver algunas dudas que no era capaz de solucionar. Hubo muchos momentos de frustación pero al final conseguí avanzar, solucionar y cumplir casi todos los objetivos que me había propuesto inicialmente. Por todo esto se han incluido tantos manuales y explicaciones de cómo poner todo en marcha, ya que es algo que me gustaría que tendrían ya solucionado en la Universidad de Pinar del Río para futuros proyectos.

- Desde el primer momento pensé que realizar el proyecto en Cuba iba a ser una tarea muy complicada pero nunca imagine que iba a ser tan sufrido. A pesar de todo fue un reto y una satisfacción única que ha merecido la pena. Un esfuerzo del cual te sientes realmente orgulloso y realizado. Sobre todo me quedo con el cariño y la ayuda recibida por parte no solo de los profesores sino también de las personas externas a la universidad que me ayudaron a realizar el proyecto. Incluso estaban conmigo los domingos ya que durante la semana trabajaban, con una alegría y una satisfacción por ayudarme inmensa. Lo que más me ha llenado de esta experiencia no solo ha sido la capacidad que tiene la gente para solventar problemas sin ningún recurso sino también la cercanía y cariño que demuestran. La bondad, la amistad, la solidaridad, siempre te ofrecían lo poco que tenían o sabían.

8.2. Líneas futuras

Este proyecto deja infinidad de posibilidades de ampliación en el que se pueden abrir varías líneas futuras de investigación y desarrollo.
Android es una plataforma de desarrollo en crecimiento exponencial en la que continuamente aparecen funcionalidades nuevas y mejoradas. Las posibilidades se ampliaran pudiendo desarrollar aplicaciones más completas. Algunas de estas líneas futuras de investigación pueden ser:

- Implementar el envío de comandos de Android a Arduino. Habilitar los botones btnOn y btnOff para que tengan una funcionalidad distinta de la de encender o apagar un led. Por ejemplo habilitar o deshabilitar la conexión bluetooth.

- Implementar el envío de comandos de Android a Arduino mediante la opción *menú* disponible en la clase *MainActivity.java.*
  Con la opción del menú *Muestreo,* tener la opción de seleccionar el tiempo de refresco con el que se reciben los datos de los sensores de Arduino.
  Con la opción del menú *Escala*, tener la opción de modificar la escala de las gráficas de los datos recibidos de los sensores.
  Con la opción del menú *Variables,* tener la opción de seleccionar las variables deseadas. Encender o apagar las variables…

9. Bibliografía

http://developer.android.com/training/index.html

http://developer.android.com/guide/topics/connectivity/bluetooth.html

http://www.androidcurso.com/

http://www.sgoliver.net/

http://www.arduino.cc/es/

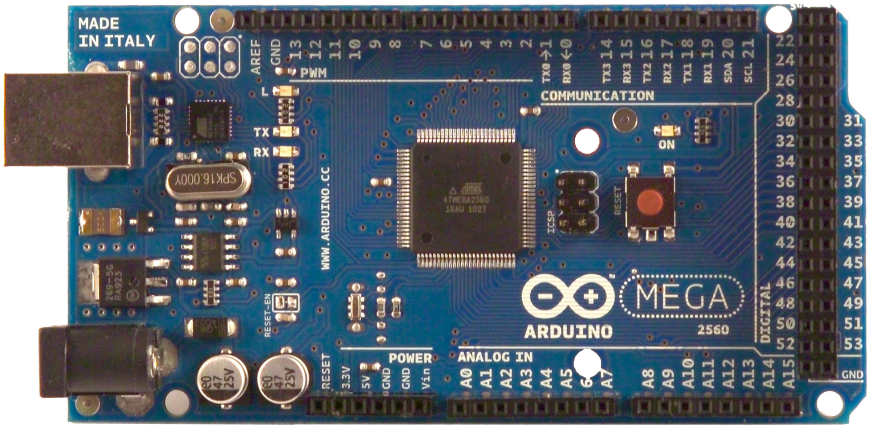
http://arduino.cc/en/Main/ArduinoBoardMega2560

Desarrollo de aplicaciones Android seguras (Miguel Ángel Moreno) - Edición Informática 64

10. ANEXO I

Características técnicas de la placa Arduino Mega 2560

Arduino MEGA 2560



CE

Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

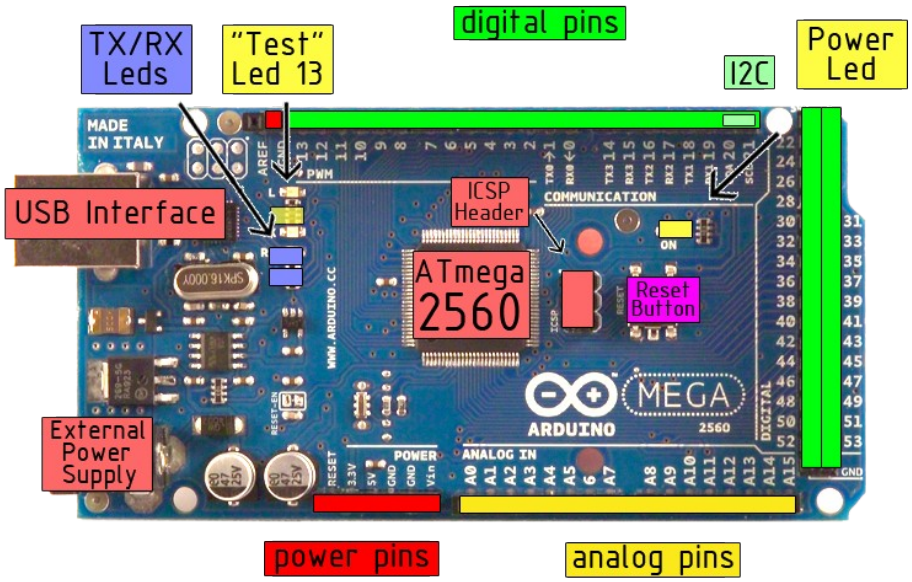
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

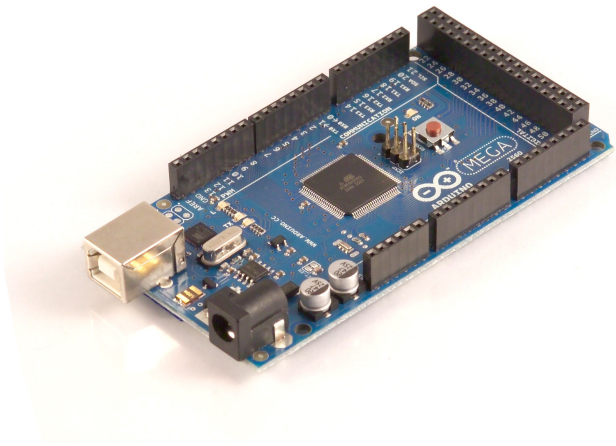
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install Windows Install Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

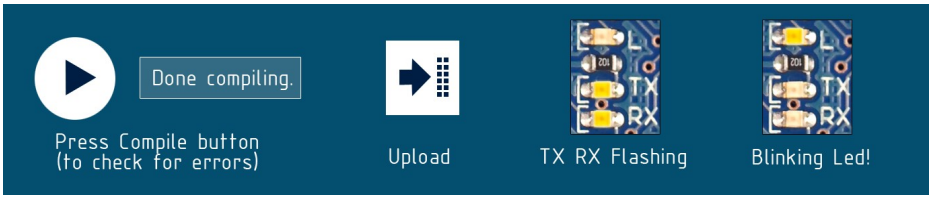
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

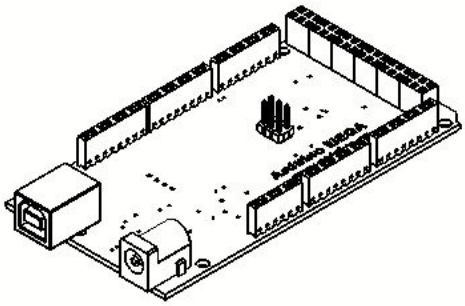
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

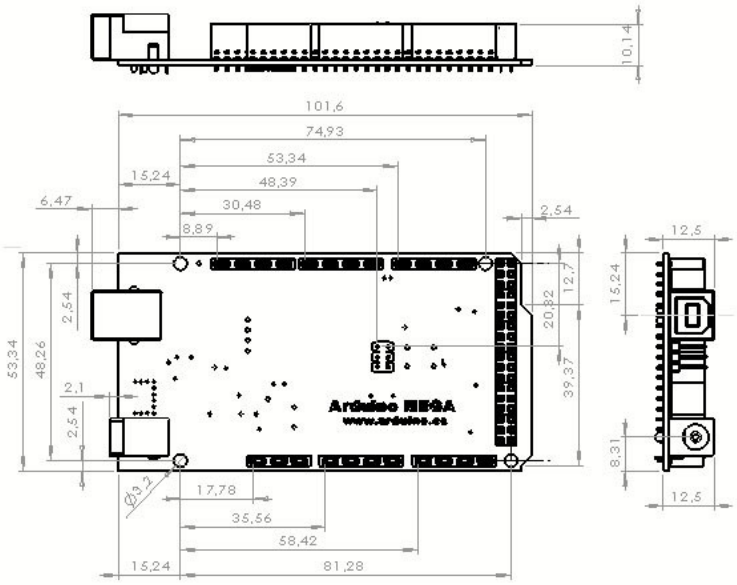
Once you have your skecth you'll see something very close to the screenshot on the right.

In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.







Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

11. ANEXO II

Características técnicas del módulo bluetooth LM780

LM TECHNOLOGIES®
INNOVATIVE TECHNOLOGY PRODUCTS

LM780 Bluetooth Serial Data Module
25m Distance with On-Board Antenna

Product: LM780
Part No: 780-0220 for BT2.0 F/W
Part No: 780-0223 for BT2.1 F/W
Datasheet Rev1.2/19-04-12

1 General Description

LM780 is LM Technologies Ltd Class 2 Bluetooth Data module with on-board chip antenna. This module is a CSR Bluecore 4 (BC04) chip based surface mount module available with Bluetooth Serial Port Profile (SPP) firmware. This module is ideal for adding short range wireless connectivity to embedded products. The module acts as a standalone unit (i.e. it does not need a host to drive it) and can interface with embedded microcontrollers via UART. It operates over a wide voltage range of 3.3V to 5V and gives excellent performance over a distance of 10-25 m.

This module is available with Bluetooth 2.0+EDR as well as Bluetooth 2.1+EDR compliant SPP firmware. The module also comes with Bluetooth 2.1 + EDR compliant HID Keyboard firmware which is available upon request.

2 Features

- Bluetooth v2.0 + EDR and v2.1 + EDR compliant firmware available
- Class 2 radio with integrated chip antenna - 25 m range in open space
- Low Power consumption
- Secure Simple Pairing supported (in Bluetooth 2.1 + EDR firmware)
- 3V - 5.5V operation
- Full Bluetooth EDR data rate of upto 3 Mbps supported
- Interface : UART (upto 921600 bps), PIO
- Multipoint firmware support
- SPP firmware supported by default. HID firmware available upon request
- CSR Bluecore 04 (BC04) chipset
- AT Command set provided for module configuration
- 802.11 Coexistence supported
- Lead free - RoHS compliant
- Small Size : 15.24 mm x 27.67 mm x 3.2mm

3 Applications

- Serial Communications
- Medical Devices
- Domestic and Industrial Applications
- Embedded Devices
- Remote Monitoring and Control
- Payment Terminals
- GPS, POS, Barcode Readers

4 Packaging Options

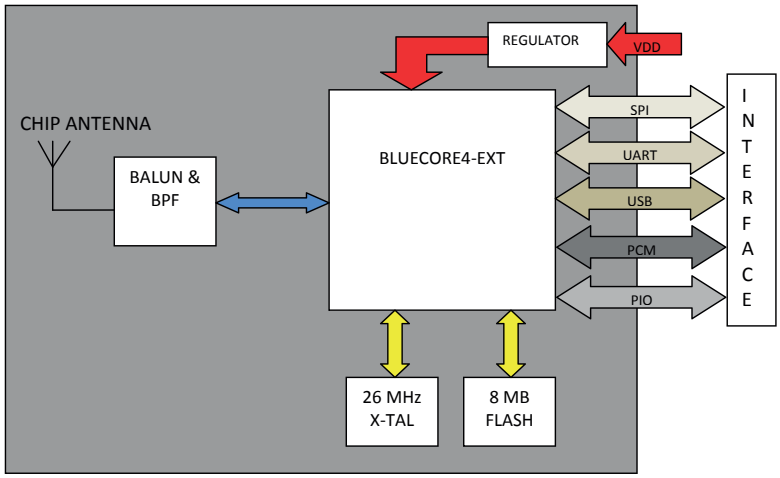
Tape and Reel
Part No 780-0222
LM780 with BT2.0 + EDR firmware

Part No 780-0225
LM780 with BT2.1 + EDR firmware

Tray Packaging
Part No 780-0221
LM780 with BT2.0 + EDR firmware

Part No 780-0224
LM780 with BT2.1 + EDR firmware

5 Block Diagram



Note: USB and PCM interface are not handled by LM780 firmware at present

6 Electrical Characteristics

Recommended Operating Conditions			
Parameter	**Min**	**Max**	**Unit**
Storage Temperature	-10	+70	°C
Supply Voltage (VDD)	+3	+5.5	V
UART pins : Tx, Rx, RTS and CTS	-0.5	+5.5	V
All other pins	VSS – 0.4	+3.3	V

Absolute Maximum Ratings			
Parameter	**Min**	**Max**	**Unit**
Storage Temperature	-40	+150	°C
Supply Voltage (VDD)	-0.3	+6.5	V
UART pins : Tx, Rx, RTS and CTS	-0.5	+7.0	V
All other pins	VSS – 0.4	+3.3	V

General Electrical Specification				
Parameter	**Description**	**Min**	**Max**	**Unit**
Input Low Voltage	RESET, PIO, PCM	-0.3	+0.8	V
Input Low Voltage	UART		0.3x VDD	V
Input High Voltage	RESET, PIO, PCM	0.7 x VDD	VDD + 0.3	V
Input High Voltage	UART	0.7 x VDD		V

7 Power Consumption Characteristics

Current Consumption		
Operation Mode	**Average**	**Unit**
Slave mode, Unconnected Idle State	19	mA
Master mode, Unconnected Idle State	5.3**	mA
Connected State, no data transfer (master and slave mode)	19	mA
Unidirectional data traffic	27	mA
Bidirectional data traffic	29	mA
Low Power Sleep Mode	0.9	mA

Input Voltage: 3.3V*
UART Data rate: 19200 bps
Firmware: LM SPP v6.13

* : Increasing power supply voltage to 5V has negligible effect on power consumption figure
** : When switching role from Master to Slave, the current consumption goes upto 37mA and then falls to 5.3mA after about 15 seconds

8 Factory Settings

The factory settings of the COM Port are as follows:

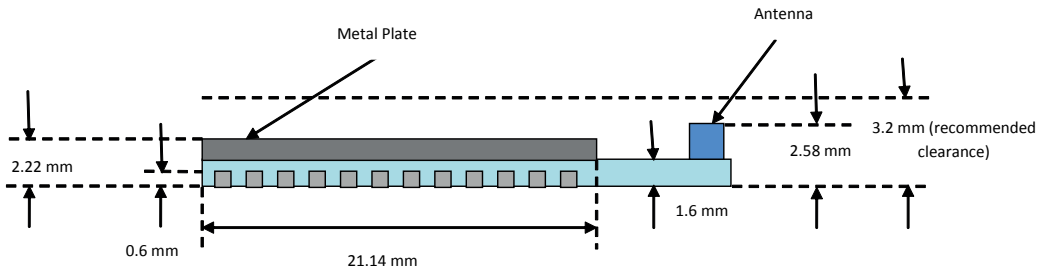
Baud Rate: 19200 bps
Data Bits: 8
Parity: None
Stop Bits: 1
Flow Control: Enabled (Hardware)

Customized settings are available as factory settings upon special request;
Bonding, Pre-Pairing, different Baud Rates, Data Bits, Parity and Flow Control Settings

9 Pin Assignments

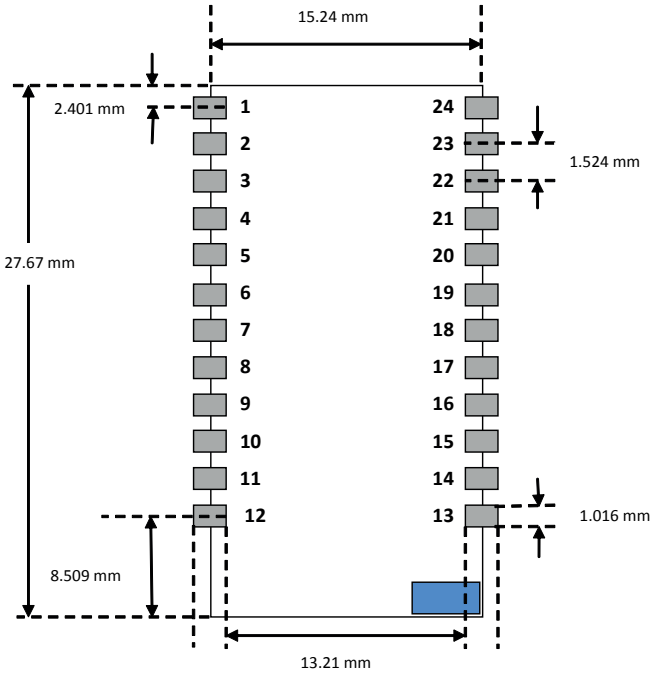
Pin Number	Name	Type	Description
1	SPI_CLK	I	SPI Clock
2	PIO 0	I/O	Programmable Input Output
3	RESET	I	Active Low Reset
4	PCM_IN	I	Synchronous Data Input
5	PCM_OUT	O	Synchronous Data Output
6	UART_TX	O	UART Data Output
7	PCM_CLK	I/O	Synchronous Data Clock
8	UART_RX	I	UART Data Input
9	UART_CTS	I	UART Clear to Send (Active Low)
10	UART_RTS	O	UART Request to Send (Active Low)
11	USB_DN	I/O	USB Data Minus
12	USB_DP	I/O	USB Data Plus
13	PCM_SYNC	I/O	Synchronous Data Sync
14	PIO 7	I/O	Programmable Input Output
15	PIO 6	I/O	Programmable Input Output
16	SPI_MISO	O	SPI Data Output
17	SPI_MOSI	I	SPI Data Input
18	PIO 1	I/O	Programmable Input Output
19	SPI_CS	I	Chip Select for SPI Interface
20	PIO 2	I/O	Programmable Input Output
21	PIO 8	I/O	Programmable Input Output
22	PIO 3	I/O	Programmable Input Output
23	VSS	N/A	Ground
24	VDD	I	Power Supply

10 Dimensions



Side view*

***: Note- Shown Dimensions not as per scale**



Module dimension and suggested landing pattern*

***: Note- Shown Dimensions not as per scale**

11 Revision History

Revision	Date	Comments
v1.0	17/12/2010	First Version
v1.1	03/04/2012	1. Added metal cap to the module 2. Module dimensions changed 3. Added part number information 4. Added packaging options
v1.2	19/04/2012	1. Added antenna outline to landing pattern diagram 2. Added Revision History

LM780 Module Integration Notes
Document Version: 1.2

Version History

Version	Date	Description
v1.0	07/06/2010	First Version
v1.1	22/10/2012	1. Add information about pin connections 2. Add information about Reset Circuit
v1.2	13/03/2013	1. Update PIO information for Ring Indicator, Carrier Detect and other functions

Contents

1 General Guidelines -Module Positioning

LM780 module includes on-board antenna and RF tracks on the PCB board which necessitates the customer board/end product hardware designer to follow recommended guidelines during placement of LM780 modules in customer product.

The below guidelines must be considered for positioning LM780 module:

- Components should be at least 5mm away from the module, this will help to maximise the range achievable
- There should be no planes directly underneath the module
- The number of PCB traces underneath the module should be kept to a minimum
- All unused pins can be left disconnected
- Do not use ultrasonic cleaning as this can cause damage to the solder connections

2 Pin Connections

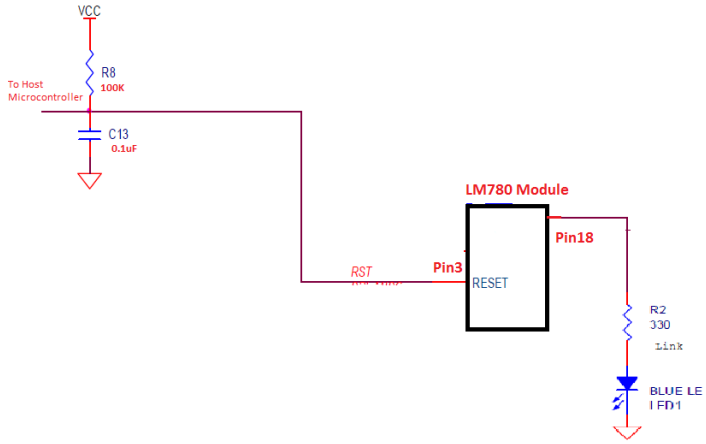
The following table shows the minimum number of pin connections and their functions to make the module usable

Functionality	LM780 Module Pin	Host Processor Signal	Remarks
Link Indication	18 (PIO 1)	Connect to LED if required	Connection status indication as shown in the Reset and LED circuit below. Blinking in slave mode, waiting for connection. In connected mode LED is solid ON
Factory Restore Button	22 (PIO 3)	Connect to button if required	Input signal. Used to restore factory setting if active high >3 sec
RF Output	NA	NA	Not required since Antenna is Onboard
3V3 (Vdd)	24	Host Power Supply (3.3V)	
GND (Vss)	23	Common Ground	
RESET	3	Pull up or connect to Host	Active Low. Refer to Reset and LED circuit below
UART_CTS	9	Host UART RTS	Short to UART_RTS(pin 10) if host doesn't support RTS/CTS flow control
UART_RTS	10	Host UART CTS	Short to UART_CTS(pin 9) if host doesn't support RTS/CTS flow control

UART_Tx	6	Host UART Rx	
UART_Rx	8	Host UART Tx	
SPI_MISO	16		Solder pad on main PCB
SPI_MOSI	17		Solder pad on main PCB
SPI_CSB	19		Solder pad on main PCB
SPI_CLK	1		Solder pad on main PCB
RI	20 (PIO 2)	Ring Indication	This pin works similar to a modem Ring Indication. Firmware pulls this PIO high by default but when there is an incoming connection request, this pin is pulled low and again high when the connection is accepted or rejected. **Please note this PIO is only used when RICD setting is enabled i.e. AT+RICD?\r=RICD+ and when module is in slave mode**
DCD	15 (PIO 6)	Data Carrier Detect	This pin works similar to Data Carrier Detect (DCD) of a modem. The firmware informs the host using this pin whether a Bluetooth connection is present or not. Firmware keeps this pin high by default but pulls it low when Bluetooth connection is present. **Please note this PIO is only used when RICD setting is enabled i.e. AT+RICD?\r=RICD+**
DTR	14 (PIO 7)	Data Terminal Ready	**This feature is only present in firmware v6.18 onwards**. It emulates the RS232 DTR pin depending on the modem signal setting (AT+MODEM command). If the modem signal setting is set to remote handshake, this signal is propagated to remote Bluetooth device depending on the status of Bluetooth connection. See AT+MODEM command description in AT Command Manual for more information.
DSR	21 (PIO 8)	Data Set Ready	**This feature is only present in firmware v6.18 onwards**. It emulates the RS232 DSR pin depending on the modem signal setting (AT+MODEM command). This pin should be connected to the host DTR. If the modem signal setting is set to remote handshake, this signal is propagated to remote Bluetooth device. So the remote

			device will know the status of local host's DTR pin. See AT+MODEM command description in AT Command Manual for more information.

The Reset and LED Circuit of the LM780 module is shown below. A push button can be included in the reset circuit to pull the Reset PIN3 low to reset the module.



Reset and LED Circuit

12. ANEXO III

Código XML implementado

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="#4B088A" >

    <Button
        android:id="@+id/btnOn"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="ON"
        android:textColor="#ff8000"/>

    <Button
        android:id="@+id/btnOff"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/btnOn"
        android:layout_toRightOf="@+id/btnOn"
        android:text="OFF"
        android:textColor="#ff8000"/>

    <Button
        android:id="@+id/btnPlot"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/btnOff"
        android:layout_toRightOf="@+id/btnOff"
        android:text="GRÁFICAS"
        android:textColor="#ff8000"/>

    <SeekBar
        android:id="@+id/seek1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/btnOn"
        android:layout_marginTop="14dp"
        android:layout_toLeftOf="@+id/textView1"
        android:indeterminate="false"
        android:max="1024" />
```

```
<SeekBar
    android:id="@+id/seek2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignRight="@+id/seek1"
    android:layout_below="@+id/seek1"
    android:layout_marginTop="14dp"
    android:indeterminate="false"
    android:max="1024" />

 <SeekBar
     android:id="@+id/seek3"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_alignRight="@+id/seek2"
     android:layout_below="@+id/seek2"
     android:layout_marginTop="14dp"
     android:indeterminate="false"
     android:max="1024" />

<SeekBar
    android:id="@+id/seek4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignRight="@+id/seek3"
    android:layout_below="@+id/seek3"
    android:layout_marginTop="14dp"
    android:indeterminate="false"
    android:max="1024" />

 <SeekBar
     android:id="@+id/seek5"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_alignRight="@+id/seek4"
     android:layout_below="@+id/seek4"
     android:layout_marginTop="14dp"
     android:indeterminate="false"
     android:max="1024" />

<SeekBar
    android:id="@+id/seek6"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignRight="@+id/seek5"
    android:layout_below="@+id/seek5"
    android:layout_marginTop="14dp"
    android:indeterminate="false"
    android:max="1024" />

 <SeekBar
     android:id="@+id/seek7"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignRight="@+id/seek6"
        android:layout_below="@+id/seek6"
        android:layout_marginTop="14dp"
        android:indeterminate="false"
        android:max="1024" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignTop="@+id/seek2"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView3"
        android:layout_alignTop="@+id/seek4"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

    <TextView
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView4"
        android:layout_alignTop="@+id/seek5"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView5"
        android:layout_alignTop="@+id/seek6"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

    <TextView
        android:id="@+id/textView7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView6"
        android:layout_alignTop="@+id/seek7"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />
```

```
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView2"
        android:layout_alignTop="@+id/seek3"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/seek1"
        android:text="Small Text"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffffff" />

</RelativeLayout>
```

grafico_layout.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
     >
    <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="250dip"
    android:orientation="vertical"
    android:id="@+id/grafico_id"
     />


    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/datatext"
        android:text=""

        />
</LinearLayout>
```

plot_items.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:src="@drawable/ic_launcher"
        />
    <TextView
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:id="@+id/item_seek"
         android:textColor="#ffffff"
        />

</LinearLayout>
```

plot_layout.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:background="#4B088A" >
   <ListView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:id="@+id/list"
       android:textColor="#ffffff">

   </ListView>

</LinearLayout>
```

escala.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >


</LinearLayout>
```

muestreo.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Introduzca tiempo de muestreo:" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"/>

    <Button
        android:id="@+id/btnEnviar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enviar" />

</LinearLayout>
```

variables.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >


</LinearLayout>
```

AndroidManifest.xml:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.proyectofinal8"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
      <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:name="com.example.proyectofinal8.Compartirdatos"
        >
        <activity
            android:name=".MainActivity"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            android:screenOrientation="portrait"
            >
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
         <activity
            android:name=".PlotActivity"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            >


        </activity>
         <activity
            android:name=".test"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            android:screenOrientation="landscape"
            >

        </activity>
            <activity
            android:name=".Muestreo"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            >

        </activity>

            <activity
            android:name=".Escala"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            >

        </activity>

             <activity
            android:name=".Variables"
            android:label="proyectofinal8"

android:configChanges="orientation|screenSize|locale|keyboardHidden|keyboard"
            >

        </activity>
    </application>

</manifest>
```

13. ANEXO IV

Código JAVA implementado

Compartirdatos.java:

```
package com.example.proyectofinal8;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.UUID;



import android.app.Application;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Build;
import android.os.Handler;
import android.util.Log;

public class Compartirdatos extends Application {

	final int RECIEVE_MESSAGE = 1;
	static BluetoothAdapter btAdapter = null;
	static BluetoothSocket btSocket = null;
	static StringBuilder sb = new StringBuilder();
	static Compartirdatos compartir;

	static ConnectedThread mConnectedThread;

	// Servicio SPP UUID de Arduino
	private static final UUID MY_UUID = UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB");

	// Dirección MAC del módulo bluetooth LM780
	private static String address = "00:12:6F:26:6D:74";

	static ArrayList<ArrayList<Integer>> datos;

      @Override
      public void onCreate() {
            // TODO Auto-generated method stub
            super.onCreate();
            datos = new ArrayList<ArrayList<Integer>>();//Array de los datos
obtenidos
            for (int i = 0; i < 7; i++)
            {
              datos.add(new ArrayList<Integer>());
            }

            btAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
            CreateConnection();
            mConnectedThread = new ConnectedThread(btSocket);

            CreateConnection();
            compartir = this;

      }

      public static void ReiniciarDatos()
      {
            datos = new ArrayList<ArrayList<Integer>>();
            for (int i = 0; i < 7; i++)
            {
              datos.add(new ArrayList<Integer>());
            }
      }

      private void checkBTState() {
          // Comprobar la compatibilidad de Bluetooth y después comprobar para
asegurarse de que está encendido
          // El emulador no soporta Bluetooth y devolverá "null"
          if(btAdapter==null) {

          } else {
            if (btAdapter.isEnabled()) {

            } else {
              // Solicitar al usuario que active la opción Bluetooth
              Intent            enableBtIntent            =            new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);

            }
          }
        }

      public static Compartirdatos GetInstance()
      {
            return compartir;
      }

        public static void CreateConnection()
        {
              BluetoothDevice device = btAdapter.getRemoteDevice(address);

                // Se necesitan dos cosas para hacer una conexión:
                // Una dirección MAC, dada anteriormente.
                // Un ID de servicio o UUID. En este caso estamos usando el
UUID para SPP.



              if(Build.VERSION.SDK_INT >= 10){
                try {
                    final           Method                        m           =
device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord",    new
Class[] { UUID.class });
                    btSocket =  (BluetoothSocket) m.invoke(device, MY_UUID);
                } catch (Exception e) {
```

```
                }
            }
               else
               {
                     try {
                          btSocket                                                =
device.createRfcommSocketToServiceRecord(MY_UUID);
                     } catch (IOException e) {
                          // TODO Auto-generated catch block
                          e.printStackTrace();
                     }
               }


                  try {
                  btSocket                                                =
device.createRfcommSocketToServiceRecord(MY_UUID);
                } catch (IOException e) {

                }

                  // El descubrimiento del dispositivo es intensivo en
recursos. Hay que asegúrarse de que no está pasando
                  // Cuando se intenta conectar y transmitir su mensaje.
                btAdapter.cancelDiscovery();

                // Establecer la conexión. Esto se bloqueará hasta que se
conecte.

                try {
                  btSocket.connect();

                } catch (IOException e) {
                  try {
                    btSocket.close();
                  } catch (IOException e2) {

                  }
                }

                // Se Crea un flujo de datos para que podamos hablar con el
servidor.

        }

        private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
throws IOException {
            if(Build.VERSION.SDK_INT >= 10){
                try {
                    final           Method                          m           =
device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord",    new
Class[] { UUID.class });
                    return (BluetoothSocket) m.invoke(device, MY_UUID);
                } catch (Exception e) {

                }
            }
```

```
            return  device.createRfcommSocketToServiceRecord(MY_UUID);
        }

}
```

ConnectedThread.java:

```
package com.example.proyectofinal8;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import android.bluetooth.BluetoothSocket;
import android.os.Handler;
import android.util.Log;

public class ConnectedThread extends Thread {

           private final InputStream mmInStream;
           private final OutputStream mmOutStream;
           Handler h;

           public ConnectedThread(BluetoothSocket socket) {
               InputStream tmpIn = null;
               OutputStream tmpOut = null;

               // Obtener los flujos de entrada y de salida, y el uso de
objetos temporales
               try {
                   tmpIn = socket.getInputStream();
                   tmpOut = socket.getOutputStream();
               } catch (IOException e) { }

               mmInStream = tmpIn;
               mmOutStream = tmpOut;
           }

           public void run() {
               byte[] buffer = new byte[256];  // Creamos un buffer de bytes
               int bytes; // Recibimos los bytes

               // Sigue escuchando el InputStream hasta que se produce una
excepción
               while (true) {
                    try {
                       // Leer desde el InputStream
                       bytes = mmInStream.read(buffer);// Obtener el número de
bytes y el mensaje de "buffer"
                  h.obtainMessage(1,   bytes,   -1,   buffer).sendToTarget();//
Enviar el mensaje a la cola del Handler
                   } catch (IOException e) {
                       break;
                   }
```

```
            }
        }

        // Llamar a esta parte del código desde la actividad principal para
enviar datos al dispositivo remoto.
        public void write(String message) {
          byte[] msgBuffer = message.getBytes();
          try {
               mmOutStream.write(msgBuffer);
           } catch (IOException e) {

             }
        }

        public void SetHandler(Handler handler)
        {
          h = handler;
        }

}
```

MainActivity.java:

```
package com.example.proyectofinal8;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.UUID;

import com.example.proyectofinal8.R;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

@SuppressLint("HandlerLeak")
public class MainActivity extends Activity {
  private static final String TAG = "proyectofinal8";
```

```
  Button btnOn, btnOff, btnPlot;
  TextView txtArduino;
  TextView txtDatos;
  SeekBar seek1;
  SeekBar seek2;
  SeekBar seek3;
  SeekBar seek4;
  SeekBar seek5;
  SeekBar seek6;
  SeekBar seek7;
  TextView textView1;
  TextView textView2;
  TextView textView3;
  TextView textView4;
  TextView textView5;
  TextView textView6;
  TextView textView7;
  TextView textView8;

  Handler h;

  final int RECIEVE_MESSAGE = 1;            // Estado del Handler
  private BluetoothAdapter btAdapter = null;
  private BluetoothSocket btSocket = null;
  private StringBuilder sb = new StringBuilder();

  private ConnectedThread mConnectedThread;

  // Servicio SPP UUID de Arduino
  private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

  // Dirección MAC del módulo bluetooth LM780
  private static String address = "00:12:6F:26:6D:74";

  Compartirdatos compartir;

  ArrayList<ArrayList<Integer>> store;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);



    btnOn = (Button) findViewById(R.id.btnOn); // Botón ON
    btnOff = (Button) findViewById(R.id.btnOff); // Botón OFF
    btnPlot = (Button) findViewById(R.id.btnPlot); // Botón GRÁFICAS

    btnPlot.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                  // TODO Auto-generated method stub
                  Intent   intent   =   new   Intent(getApplicationContext(),
```

```
PlotActivity.class);

                    startActivity(intent);

            }

   });


    seek1 = (SeekBar) findViewById(R.id.seek1); // Barra progresiva variable 1
    seek2 = (SeekBar) findViewById(R.id.seek2); // Barra progresiva variable 2
    seek3 = (SeekBar) findViewById(R.id.seek3); // Barra progresiva variable 3
    seek4 = (SeekBar) findViewById(R.id.seek4); // Barra progresiva variable 4
    seek5 = (SeekBar) findViewById(R.id.seek5); // Barra progresiva variable 5
    seek6 = (SeekBar) findViewById(R.id.seek6); // Barra progresiva variable 6
    seek7 = (SeekBar) findViewById(R.id.seek7); // Barra progresiva variable 7
    textView1    =    (TextView)    findViewById(R.id.textView1);    //    Texto
correspondiente a la variable 1
    textView2    =    (TextView)    findViewById(R.id.textView2);    //    Texto
correspondiente a la variable 2
    textView3    =    (TextView)    findViewById(R.id.textView3);    //    Texto
correspondiente a la variable 3
    textView4    =    (TextView)    findViewById(R.id.textView4);    //    Texto
correspondiente a la variable 4
    textView5    =    (TextView)    findViewById(R.id.textView5);    //    Texto
correspondiente a la variable 5
    textView6    =    (TextView)    findViewById(R.id.textView6);    //    Texto
correspondiente a la variable 6
    textView7    =    (TextView)    findViewById(R.id.textView7);    //    Texto
correspondiente a la variable 7


     compartir  =  Compartirdatos.GetInstance();  //  Recive  los  datos  y  la
conexión bluetooth de la Activitity Compartirdatos
     store = compartir.datos;

       h = new Handler() {
      public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
           case RECIEVE_MESSAGE:
                                      // Si recibe el mensaje
            byte[] readBuf = (byte[]) msg.obj;

            String strIncom = new String(readBuf, 0, msg.arg1);
            // Crea una cadena de array de bytes
            proccessTrama(strIncom.getBytes());

            sb.append(strIncom);
                              // Se anexa la cadena
            int endOfLineIndex = sb.indexOf("\r\n");      //>+1
            // Determina el final de cada línea
            if (endOfLineIndex > 0) {
                              // Si es el final de línea extrae la cadena
                  String sbprint = sb.substring(0, endOfLineIndex);
            // Extrae la cadena
                   sb.delete(0, sb.length());
                              // Limpia la cadena
                  btnOff.setEnabled(true);
```

```
                    btnOn.setEnabled(true);

                }

            Log.d(TAG, "...String:"+ sb.toString() +   "Byte:" + msg.arg1 +
"...");
            break;
            }

        }

        public void proccesdata(String cadena) {
                    // TODO Auto-generated method stub

             try
             {

                            int indice=1;
                            for (int i=0; i<7;i++){
                                  lD[i]=(byte)(cadena.charAt(indice) - '0');
                                  indice=indice+1;
                                  String                  gg                  =
cadena.substring(indice,indice+lD[i]);
                                  Datos[i]= Integer.parseInt(gg);
                                  indice=indice+lD[i];
                        // Los datos que se reciben se acumulan hasta un total
de 20 datos por variable
                            // Despúes se reinicia la cadena y se vuelven a
acumular otros 20 datos
                            }
                            if(store.get(0).size() == 20)
                            {
                                  compartir.ReiniciarDatos();
                                  store = compartir.datos;

                            }
                            // Se añaden los datos a las barras progresivas
                            // Los valores de los datos se refrescan cada
segundo
                            seek1.setProgress(Datos[0]);
                            store.get(0).add(Datos[0]);

                            seek2.setProgress(Datos[1]);
                            store.get(1).add(Datos[1]);

                            seek3.setProgress(Datos[2]);
                            store.get(2).add(Datos[2]);

                            seek4.setProgress(Datos[3]);
                            store.get(3).add(Datos[3]);

                            seek5.setProgress(Datos[4]);
                            store.get(4).add(Datos[4]);

                            seek6.setProgress(Datos[5]);
                            store.get(5).add(Datos[5]);

                            seek7.setProgress(Datos[6]);
```

```
                        store.get(6).add(Datos[6]);

                        // Se añaden los datos a la Vista de texto
                        //  Los  valores  de  los  datos  se  refrescan  cada
segundo
                        textView1.setText("v1:" + Datos[0]);
                        textView2.setText("v2:" + Datos[1]);
                        textView3.setText("v3:" + Datos[2]);
                        textView4.setText("v4:" + Datos[3]);
                        textView5.setText("v5:" + Datos[4]);
                        textView6.setText("v6:" + Datos[5]);
                        textView7.setText("v7:" + Datos[6]);


                        int cont = inserted;
                        String mostrarBuffer ="";
                        for(int  pos=(write-1)%cant;  cont>0;pos=(pos-1)%cant
){
                              int[] iter = bufferDatos[pos];


                              for(int val : iter)
                              {
                                 System.out.println(val);
                                 mostrarBuffer += String.valueOf(val)+",";
                              }
                              guardarDato(Datos);

                              //En  iter  se  recorren  los  datos  recibidos
desde el ultimo hasta el primero
                        }



                        // Se añaden los datos a las barras progresivas
                        //  Los  valores  de  los  datos  se  refrescan  cada
segundo


                        seek1.setProgress(Datos[0]);
                        store.get(0).add(Datos[0]);

                        seek2.setProgress(Datos[1]);
                        store.get(1).add(Datos[1]);

                        seek3.setProgress(Datos[2]);
                        store.get(2).add(Datos[2]);

                        seek4.setProgress(Datos[3]);
                        store.get(3).add(Datos[3]);

                        seek5.setProgress(Datos[4]);
                        store.get(4).add(Datos[4]);

                        seek6.setProgress(Datos[5]);
                        store.get(5).add(Datos[5]);
```

```
                        seek7.setProgress(Datos[6]);
                        store.get(6).add(Datos[6]);

                        // Se añaden los datos a la Vista de texto
                        //  Los  valores  de  los  datos  se  refrescan  cada
segundo
                        textView1.setText("v1:" + Datos[0]);
                        textView2.setText("v2:" + Datos[1]);
                        textView3.setText("v3:" + Datos[2]);
                        textView4.setText("v4:" + Datos[3]);
                        textView5.setText("v5:" + Datos[4]);
                        textView6.setText("v6:" + Datos[5]);
                        textView7.setText("v7:" + Datos[6]);


                        String mostrar ="";
                        for(int val : Datos)
                        {
                           System.out.println(val);
                           mostrar += String.valueOf(val)+",";
                        }
                               guardarDato(Datos);

                    }catch(Exception ex){

                           }

                          }

                      //Obtener la cadena que se va a procesar para extraer los datos.
                    byte lD[]=new byte[7];
                    int Datos[]=new int[7];
                    //byte indice=1;
                    String cadena = "";

                    boolean abierto=false;
                      public void proccessTrama(byte[] readBuf) {
                            // TODO Auto-generated method stub


                          for(byte c: readBuf){
                                if(c=='<'){
                              cadena="<";
                              abierto=true;
                                }
                                else if (c=='>'){
                                       cadena +=(char)c;
                                       if (abierto){
                                       proccesdata(cadena);}
                                       abierto=false;
                                }
                                else {cadena +=(char)c;}

                          }
                    }
            };

            btAdapter = compartir.btAdapter;
```

```
      // Conseguir Bluetooth adapter
    checkBTState();
    compartir.mConnectedThread.SetHandler(h);
    compartir.mConnectedThread.start();


    btnOn.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
       btnOn.setEnabled(false);
       mConnectedThread.write("1");     // Enviar "1" via Bluetooth
        Toast.makeText(getBaseContext(),         "Turn         on         LED",
Toast.LENGTH_SHORT).show();
      }
    });

    btnOff.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
       btnOff.setEnabled(false);
       mConnectedThread.write("0");     // Enviar "0" via Bluetooth
        Toast.makeText(getBaseContext(),         "Turn         off         LED",
Toast.LENGTH_SHORT).show();
      }
    });
  }




  //Buffer de datos para la Gráfica
  final int cant = 6;
  int write = 0;
  int inserted = 0;

  int[][] bufferDatos = new  int[cant][];
  protected void guardarDato(int[] datos) {
       // TODO Auto-generated method stub
        bufferDatos[write] = datos.clone();
        write = (write+1) % cant;
        if(inserted<cant)
        inserted++;

  }




  private    BluetoothSocket    createBluetoothSocket(BluetoothDevice    device)
throws IOException {
      if(Build.VERSION.SDK_INT >= 10){
          try {
              final             Method                              m             =
device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord",    new
Class[] { UUID.class });
              return (BluetoothSocket) m.invoke(device, MY_UUID);
          } catch (Exception e) {
              Log.e(TAG, "Could not create Insecure RFComm Connection",e);
```

```
            }
        }
        return  device.createRfcommSocketToServiceRecord(MY_UUID);
    }

    @Override
    public void onResume() {
      super.onResume();


    }

    @Override
    public void onPause() {
      super.onPause();

    }

    private void checkBTState() {
    // Comprobar la compatibilidad de Bluetooth para asegurarse de que está
encendido
    // El emulador no soporta Bluetooth y devolverá "null"
      if(btAdapter==null) {
        errorExit("Fatal Error", "Bluetooth not support");
      } else {
        if (btAdapter.isEnabled()) {
          Log.d(TAG, "...Bluetooth ON...");
        } else {
          // Solicitar al usuario que active la opción Bluetooth
          Intent                enableBtIntent                =                new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
          startActivityForResult(enableBtIntent, 1);
        }
      }
    }

    private void errorExit(String title, String message){
      Toast.makeText(getBaseContext(),    title    +    "    -    "    +    message,
Toast.LENGTH_LONG).show();
      finish();
    }

    public class ConnectedThread extends Thread {
            private final InputStream mmInStream;
            private final OutputStream mmOutStream;
            Handler h;

            public ConnectedThread(BluetoothSocket socket, Handler handler) {
                InputStream tmpIn = null;
                OutputStream tmpOut = null;
                h = handler;

                // Recibe los flujos de entrada y de salida y el uso de objetos
temporales
                try {
                    tmpIn = socket.getInputStream();
                    tmpOut = socket.getOutputStream();
                } catch (IOException e) { }
```

```
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[256];  // Creamos un buffer de bytes
            int bytes; // Recibimos los bytes


            // Sigue escuchando el InputStream hasta que se produce una
excepción
            while (true) {
                 try {
                        // Leer desde el InputStream
                    bytes = mmInStream.read(buffer); // Obtener el número de
bytes y el mensaje de "buffer"
                     h.obtainMessage(RECIEVE_MESSAGE,         bytes,         -1,
buffer).sendToTarget(); // Enviar el mensaje a la cola del Handler
                  } catch (IOException e) {
                     break;
                  }
            }
        }

        // Llamar a esta parte del código desde la actividad principal para
enviar datos al dispositivo remoto.
        public void write(String message) {
           Log.d(TAG, "...Data to send: " + message + "...");
           byte[] msgBuffer = message.getBytes();
           try {
                mmOutStream.write(msgBuffer);
            } catch (IOException e) {
                Log.d(TAG, "...Error data send: " + e.getMessage() + "...");
              }
        }


  }

  @Override


   public boolean onCreateOptionsMenu(Menu menu) {
                  // Agrega elementos a la barra de acción, si está presente.
                  getMenuInflater().inflate(R.menu.main, menu);
                  return true;
        }

       @Override
       public boolean onOptionsItemSelected(MenuItem Muestreo){

             Intent intent = new Intent();

              switch (Muestreo.getItemId()) {
                case R.id.Muestreo:
                       intent.setClass(getApplicationContext(),
Muestreo.class);
```

```
                    break;
            case R.id.Escala:
              intent.setClass(getApplicationContext(), Escala.class);
                    break;
            case R.id.Variables:
              intent.setClass(getApplicationContext(), Variables.class);
                 break;

                  default:
                        break;
                  }

                  startActivity(intent);
                  return false;
          }

          public boolean onCreateOptionsMenu1(Menu menu) {
                 // Agrega elementos a la barra de acción, si está presente.
                  getMenuInflater().inflate(R.menu.main, menu);
                  return true;
           }

          public boolean onOptionsItemSelected1(MenuItem Escala){


                  Intent              intent              =              new
Intent(getApplicationContext(),Escala.class);
                  startActivity(intent);
                  return false;
          }

          public boolean onCreateOptionsMenu11(Menu menu) {
                 // Agrega elementos a la barra de acción, si está presente.
                  getMenuInflater().inflate(R.menu.main, menu);
                  return true;
           }

          public boolean onOptionsItemSelected11(MenuItem Variables){
                 Intent              intent              =              new
Intent(getApplicationContext(),Variables.class);
                  startActivity(intent);
                  return false;
          }

}
```

PlotActivity.java:

```
package com.example.proyectofinal8;

import java.util.ArrayList;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
```

```
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.Toast;

public class PlotActivity extends Activity
{
     @Override
    protected void onCreate(Bundle savedInstanceState) {
      // TODO Auto-generated method stub
      super.onCreate(savedInstanceState);
      setContentView(R.layout.plot_layout);
      ListView list = (ListView)findViewById(R.id.list);
      ArrayList<String> _values = new ArrayList<String>();

      for (int i = 1; i < 8; i++) {
                  _values.add("GRÁFICA EN TIEMPO REAL: "+i);
            }
      final  plotAdapter  adapter  =  new  plotAdapter(getLayoutInflater(),
_values);
      list.setAdapter(adapter);

      list.setOnItemClickListener(new OnItemClickListener() {

                  @Override
                  public  void  onItemClick(AdapterView<?>  arg0,  View  arg1,
int arg2,
                              long arg3) {
                        // TODO Auto-generated method stub
                        Intent intent = new Intent(getApplicationContext(),
test.class);
                        intent.putExtra("name",adapter.values.get(arg2) );
                        intent.putExtra("index",arg2);

                        startActivity(intent);

                  }
            });

    }
}
```

plotAdapter.java:

```
package com.example.proyectofinal8;

import java.util.ArrayList;

import com.example.proyectofinal8.R.layout;

import android.R.integer;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.SeekBar;
```

```
import android.widget.TextView;

public class plotAdapter extends BaseAdapter {

	ArrayList<String> values;
	LayoutInflater inflater;

	public plotAdapter(LayoutInflater _inflater, ArrayList<String> _values)
	{
		values = _values;
		inflater = _inflater;
	}
	@Override
	public int getCount() {
		// TODO Auto-generated method stub
		return values.size();
	}

	@Override
	public Object getItem(int arg0) {
		// TODO Auto-generated method stub
		return values.get(arg0);
	}

	@Override
	public long getItemId(int arg0) {
		// TODO Auto-generated method stub

		return arg0;
	}

	@Override
	public View getView(int arg0, View arg1, ViewGroup arg2)
	{
		// TODO Auto-generated method stub
		View vi = arg1;
		if(vi == null)
			vi = inflater.inflate(R.layout.plot_items, null);


		TextView textview = (TextView)vi.findViewById(R.id.item_seek);
		textview.setText(values.get(arg0));

		return vi;
	}

}
```

test.java:

```
package com.example.proyectofinal8;

import android.app.Activity;
import android.content.Intent;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;
```

```
import java.util.ArrayList;
import java.util.Vector;

import com.example.proyectofinal8.R;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.Rect;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.graphics.Bitmap.Config;
import android.graphics.Paint.FontMetrics;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.ImageView;

import com.jjoe64.*;
import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphView.LegendAlign;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;




public class test extends Activity {

	// Recive los datos y la conexión bluetooth de la Activitity
Compartirdatos
	Compartirdatos compartir;
    ArrayList<Integer> datos;
    double graph2LastXValue;
    GraphViewSeries exampleSeries;
    TextView datatext;
    Handler handler = new Handler(){

	@Override
	public void handleMessage(Message msg) {
		// TODO Auto-generated method stub

		super.handleMessage(msg);
	}

    };
    double datalenght = 0;

	// Fin Compartirdatos
```

```
  // Esta etiqueta tiene los siguientes elementos[label,maxX,maxY]
static int draw_only_this_idx = -1;
static int[] drawSizes;
int index;

@Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.grafico_layout);
    setTitle("Gráfica");

    compartir = Compartirdatos.GetInstance();
    Intent intent = getIntent();
    index = intent.getIntExtra("index", -1);
    datos = compartir.datos.get(index);

    String data1 ="";
    long lenght = compartir.datos.get(index).size();
    datalenght = compartir.datos.get(index).size();
    for (int i = 0; i < lenght; i++) {
                data1+= compartir.datos.get(index).get(i)+" , ";
        }

    final int num = compartir.datos.get(index).size();
    GraphViewData[] data = new GraphViewData[num];
    double v=0;
    for (int i=0; i<num; i++) {
       v += 0.2;
       data[i] = new GraphViewData(i,compartir.datos.get(index).get(i));
    }

    graph2LastXValue = 14;

    exampleSeries = new GraphViewSeries(data);

    GraphView graphView = new LineGraphView(
               this
               , "Gráfica en tiempo real" // Título
        );

    graphView.addSeries(exampleSeries);

    // Desplazamiento y zoom
    graphView.setViewPort(0, 10);
    graphView.setScrollable(true);
    // Opcional - activar escalado / zoom
    graphView.setScalable(true);

    // Dibujando los puntos
    ((LineGraphView) graphView).setDrawDataPoints(true);
    ((LineGraphView) graphView).setDataPointsRadius(8f);

    // Dibujando el fondo
    graphView.setDrawingCacheEnabled(true);
    graphView.setBackgroundColor(Color.BLACK);

    LinearLayout layout = (LinearLayout) findViewById(R.id.grafico_id);
    layout.addView(graphView);
```

```
        Runnable runnable = new Runnable() {

                    @Override
                    public void run() {
                          // TODO Auto-generated method stub

                          double yvalue = 0;

                          {
                                yvalue                                                    =
compartir.datos.get(index).get(compartir.datos.get(index).size()-1);
                                Message msg = new Message();
                                msg.obj = yvalue+"";


                          }
                           graph2LastXValue += 1d;

                           exampleSeries.appendData(new
GraphViewData(graph2LastXValue, yvalue), true, num);
                         handler.postDelayed(this, 1000);
                    }
              };

        handler.postDelayed(runnable, 1000);


    }

    public static Bitmap quicky_XY(Bitmap bitmap)
         {
              // Codigo para obtener el mapa de bits en la pantalla
               Bitmap output = Bitmap.createBitmap(bitmap.getWidth(),
                   bitmap.getHeight(), Config.ARGB_8888);
                   Canvas canvas = new Canvas(output);

                   final int color = 0xff0B0B61;
                   final Paint paint = new Paint();
                   final  Rect  rect  =  new  Rect(0,  0,  bitmap.getWidth(),
bitmap.getHeight());
                   final RectF rectF = new RectF(rect);
                   final float roundPx = 12;

                   paint.setAntiAlias(true);
                   canvas.drawARGB(0, 0, 0, 0);
                   paint.setColor(color);
                   canvas.drawRoundRect(rectF, roundPx, roundPx, paint);


                   // Ahora sólo hay que dibujar en este mapa de bits


                   // Establecemos la información de etiquetas de forma manual
                   String[] cur_elt_array = new String[4];
                   cur_elt_array[0]="Voltage";
                   cur_elt_array[1]="volts";
                   cur_elt_array[2]="93";  // max
                   cur_elt_array[3]="0";    //min
```

```
                    Vector labels = new Vector();
                    labels.add(cur_elt_array);

                    draw_the_grid(canvas,labels);


                    // Observar que los datos pintados deberían estar en nuestro
camino

                    Vector data_2_plot = new Vector();

                    plot_array_list(canvas  ,   data_2_plot  ,   labels  , "the
title" , 0 );

                    canvas.drawBitmap(bitmap, rect, rect, paint);

                return output;
            }


    // Etiquetas de este vector[label,units,max.min]

    public static void  draw_the_grid(Canvas this_g,  Vector these_labels)
     {

        double rounded_max = 0.0;
        double rounded_min = 0.0;
        double rounded_max_temp;
        Object curElt;
        String[] cur_elt_array;
        int left_margin_d, right_margin_d;



        if( draw_only_this_idx == -1)
           curElt = these_labels.elementAt(0);
        else
           curElt = these_labels.elementAt(draw_only_this_idx);

        cur_elt_array = (String[])curElt;

        rounded_max                    =                    get_ceiling_or_floor
(Double.parseDouble(cur_elt_array[2]) , true);
        rounded_min                    =                    get_ceiling_or_floor
(Double.parseDouble(cur_elt_array[3]) , false);


       // Ahora tenemos el valor máximo de la serie
        final Paint paint = new Paint();
        paint.setTextSize(15);

       left_margin_d         =                  getCurTextLengthInPixels(paint,
Double.toString(rounded_max));
       // Mantener la posición de dibujo para más adelante
       // Dejar espacio para la leyenda
       int p_height = 170;
       int p_width = 220;
```

```
        int[] tmp_draw_sizes = {2 + left_margin_d, 25,p_width - 2 -
left_margin_d ,p_height - 25 -5};
        drawSizes = tmp_draw_sizes; // Guardarlo para su posterior
procesamiento


         // Con los márgenes elaborados dibujar la rejilla de trazado
        paint.setStyle(Paint.Style.FILL);
        paint.setColor(Color.WHITE );

        // Android dibuja por coordenadas
        this_g.drawRect(drawSizes[0], drawSizes[1],drawSizes[0]+ drawSizes[2],
drawSizes[1]+ drawSizes[3] , paint);


        paint.setColor(Color.GRAY );


         // Por último se dibuja la cuadrícula


        paint.setStyle(Paint.Style.STROKE);
        this_g.drawRect(drawSizes[0], drawSizes[1],drawSizes[0]+ drawSizes[2],
drawSizes[1]+ drawSizes[3] , paint);

            for(int i=1; i < 5 ; i++)
            {

                this_g.drawLine(drawSizes[0], drawSizes[1] + (i * drawSizes[3]
/ 5), drawSizes[0] + drawSizes[2], drawSizes[1] + (i * drawSizes[3] / 5),
paint);
                this_g.drawLine(drawSizes[0]+ (i * drawSizes[2] / 5),
drawSizes[1], drawSizes[0] + (i * drawSizes[2] / 5), drawSizes[1] +
drawSizes[3], paint);

            }



            print_axis_values_4_grid(this_g,         cur_elt_array[1]         ,
Double.toString(rounded_max) , Double.toString(rounded_min), cur_elt_array[0]
, 2 ,0 );


     } // Fin del dibujo de la cuadrícula

    public static void print_axis_values_4_grid(Canvas thisDrawingArea,
String cur_units , String cur_max , String cur_min , String cur_label , int
x_guide , int this_idx )
    {
          String this_str;
          double delta = ( Double.valueOf(cur_max).doubleValue() -
Double.valueOf(cur_min).doubleValue() ) / 5;
          final Paint paint = new Paint();

          paint.setColor( Color.WHITE );
          paint.setTypeface( Typeface.SANS_SERIF );
          paint.setTextSize(8);
```

```
          for(int i = 0; i<6 ; i++)
          {
             // Es adecuado trabajar con nuestros valores


              this_str                                              =Double.toString(
(Double.valueOf(cur_min).doubleValue()  + delta * i) );


          final int point = this_str.indexOf('.');
          if (point > 0) {
              // Si tiene un punto decimal, puede que tenga que recortar o
forzar a 2 decimales
              this_str = this_str + "00";
              this_str =  this_str.substring(0,point+3);
          } else {
              this_str = this_str + ".00";
          }


              if (i == 5)
              thisDrawingArea.drawText(this_str,   x_guide - 2, drawSizes[1] +
drawSizes[3] - (i *drawSizes[3] / 5) , paint );
              else
                  thisDrawingArea.drawText(this_str,  x_guide- 2, drawSizes[1]
+ drawSizes[3]  -  (i * drawSizes[3] / 5) -3, paint);
          }

          paint.setTextSize(10);
          switch(this_idx )
          {
           case 0:

                  thisDrawingArea.drawText("  " + cur_label +" - " +cur_units,
x_guide - 2, drawSizes[1] -15 , paint);
                  break;

           case 1:


                  thisDrawingArea.drawText("  " + cur_label +" - " +cur_units,
x_guide - 2 -30, drawSizes[1] -15, paint );
                   break;

          }



    }  // Final del eje de impresión de las 4 cuadrículas

    private static Point  scale_point(int this_x , double this_y  , Point
drawPoint ,
            int scr_x  , int scr_y  , int scr_width  , int src_height  ,
            double maxX  , double minX  , double  maxY  , double minY  )
       {
           int temp_x, temp_y;
```

```
            Point temp = new Point();

            if (maxY == minY)  // Omitir datos erroneos
                return null;

            try
            {
                    temp_x  =  scr_x  +  (int)(  ((double)this_x  -  minX)  *
((double)scr_width / (maxX - minX)) );
                    temp_y   =   scr_y   +   (int)(   (maxY   -   this_y)   *
((double)src_height / (maxY - minY)) );

                    temp.x = temp_x;
                    temp.y= temp_y;
                    drawPoint = temp;


            }
            catch  (Exception e)
            {

               return (null);
            }

            return temp;

       } // Final del punto de escala

    public   static   boolean   plot_array_list(Canvas   this_g,   Vector
this_array_list , Vector these_labels , String this_title , int only_this_idx
)
    {
              int idx;
              int lRow ;
              int nParms;
              int  i, points_2_plot, shifted_idx ;
              int prev_x, prev_y ;
              int cur_x=0, cur_y=0 ;
              // Definir el marcador como un objeto
              Point cur_point = new Point();
             cur_point.set(0,0);

              double cur_maxX, cur_minX, cur_maxY=20, cur_minY=0, cur_rangeY;
              int cur_start_x, cur_points_2_plot;

             int POINTS_TO_CHANGE = 30;
             double cur_OBD_val;

              // Objecto curElt;
              String curElt;
              String[] cur_elt_array;
              Object curElt2;
              String[] cur_elt_array2;

              final Paint paint = new Paint();

              try
              {
```

```
                    points_2_plot = this_array_list.size();
                    {
                         cur_start_x = 0;
                         cur_points_2_plot = points_2_plot;
                         cur_maxX = cur_points_2_plot;
                         cur_minX = 0;
                    }

                    // Crear los puntos de la trama de esta serie de la matriz
ChartPoints:

                    curElt = (String)this_array_list.elementAt(0);

                    // Las líneas tienen que salir bien
                     paint.setStyle(Paint.Style.STROKE);
//
                    nParms = only_this_idx;
                    {

                        // Conseguir los elementos de las etiquetas
                         curElt2 = these_labels.elementAt(nParms);
                         cur_elt_array2  = (String[]) curElt2;

                         cur_maxY            =            get_ceiling_or_floor
(Double.parseDouble(cur_elt_array2[2]) , true);
                         cur_minY            =            get_ceiling_or_floor
(Double.parseDouble(cur_elt_array2[3]) , false);

                         cur_points_2_plot = this_array_list.size();
                         cur_maxX = cur_points_2_plot;

                       curElt = (String)this_array_list.elementAt(0);
                       cur_OBD_val = Double.parseDouble( curElt);

                       cur_point = scale_point(0, cur_OBD_val, cur_point,
                               drawSizes[0],    drawSizes[1],    drawSizes[2],
drawSizes[3],
                               cur_maxX, cur_minX, cur_maxY, cur_minY);

                        cur_x = cur_point.x;
                        cur_y = cur_point.y;

                        paint.setColor(Color.RED);

                        if ( cur_points_2_plot < POINTS_TO_CHANGE)
                          this_g.drawRect(cur_x-2, cur_y-2, cur_x-2 + 4,cur_y-
2+ 4 , paint);

                        prev_x = cur_x;
                        prev_y = cur_y;

                        for (lRow = cur_start_x +1 ; lRow< cur_start_x +
cur_points_2_plot -1 ; lRow++)
                        {

                            curElt = (String)this_array_list.elementAt(lRow);
```

```
                        cur_OBD_val = Double.parseDouble( curElt);



                        // Elaborar una aproximación

                          if(  cur_OBD_val  ==  Double.NaN)  continue;    //
Saltar si es mala
                         {


                            cur_point=scale_point(lRow,        cur_OBD_val,
cur_point,
                                drawSizes[0],  drawSizes[1],  drawSizes[2],
drawSizes[3],
                                cur_maxX, cur_minX, cur_maxY, cur_minY);

                            cur_x = cur_point.x;
                            cur_y = cur_point.y;

                            if ( cur_points_2_plot < POINTS_TO_CHANGE)
                               this_g.drawRect(cur_x-2,   cur_y-2,   cur_x-2
+4, cur_y-2 + 4, paint );

                           this_g.drawLine( prev_x, prev_y, cur_x, cur_y,
paint);
                           prev_x = cur_x;
                           prev_y = cur_y;

                         }

                    } // Final para lrow

                } // Final para nParmns

            return( true);
        }
        catch (Exception e)
        {
            return( false);

        }

    } // Final de plot_array_list


    // Necesitamos la anchura de las etiquetas
    private  static  int  getCurTextLengthInPixels(Paint  this_paint,  String
this_text) {
        FontMetrics tp = this_paint.getFontMetrics();
        Rect rect = new Rect();
        this_paint.getTextBounds(this_text, 0, this_text.length(), rect);
       return rect.width();
    } // Final de getCurTextLengthInPixels


    public static double get_ceiling_or_floor(double this_val ,  boolean
```

```
is_max  )
    {
        double this_min_tmp;
        int  this_sign;
        int  this_10_factor=0;
        double this_rounded;

        if (this_val == 0.0)
        {
            this_rounded = 0.0;
            return this_rounded;
        }

        this_min_tmp = Math.abs(this_val);

        if (this_min_tmp >= 1.0 && this_min_tmp < 10.0)
            this_10_factor = 1;
        else if (this_min_tmp >= 10.0 && this_min_tmp < 100.0)
            this_10_factor = 10;
        else if (this_min_tmp >= 100.0 && this_min_tmp < 1000.0)
            this_10_factor = 100;
        else if (this_min_tmp >= 1000.0 && this_min_tmp < 10000.0)
            this_10_factor = 1000;
        else if (this_min_tmp >= 10000.0 && this_min_tmp < 100000.0)
            this_10_factor = 10000;



        // Cubrir cuando el mínimo es positivo o negativo
        if (is_max)
        {
            if (this_val > 0.0)
                this_sign = 1;
            else
                this_sign = -1;

        }
        else
        {
            if (this_val > 0.0)
                this_sign = -1;
            else
                this_sign = 1;

        }


        if (this_min_tmp > 1)
            this_rounded = (double)(((int)(this_min_tmp / this_10_factor) +
this_sign) * this_10_factor);
        else
        {
            this_rounded = (int)(this_min_tmp * 100.0);
            // cubrir igual que arriba el número bfir hasta 0.001, con menos
saltará
            if (this_rounded >= 1 && this_rounded < 9)
                this_10_factor = 1;
            else if (this_rounded >= 10 && this_rounded < 99)
```

```
                this_10_factor = 10;
            else if (this_rounded >= 100 && this_rounded < 999)
                this_10_factor = 100;

            this_rounded = (double)(((int)((this_rounded) / this_10_factor) +
this_sign) * this_10_factor);
            this_rounded = (int)(this_rounded) / 100.0;

        }

        if (this_val < 0)
          this_rounded = -this_rounded;

        return  this_rounded;

} // Final del get_ceiling_or_floor
}
```

Escala.java:

```
package com.example.proyectofinal8;

import android.app.Activity;
import android.os.Bundle;

public class Escala extends Activity {

      @Override
    protected void onCreate(Bundle savedInstanceState) {
      // TODO Auto-generated method stub
      super.onCreate(savedInstanceState);
      setContentView(R.layout.escala);
}
}
```

Muestreo.java:

```
package com.example.proyectofinal8;

import com.example.proyectofinal8.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.widget.EditText;

public class Muestreo extends Activity{
      private EditText ediciontexto;

      @Override
    protected void onCreate(Bundle savedInstanceState) {
      // TODO Auto-generated method stub
      super.onCreate(savedInstanceState);
      setContentView(R.layout.muestreo);
```

```
        this.setEdiciontexto((EditText)findViewById(R.id.editText1));
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
               // Agregar elementos a la barra de acción, si está presente.
               getMenuInflater().inflate(R.menu.main, menu);
               return true;
        }
        public EditText getEdiciontexto() {
               return ediciontexto;
        }

        public void setEdiciontexto(EditText ediciontexto) {
               this.ediciontexto = ediciontexto;



        }

}
```

Variables.java:

```
package com.example.proyectofinal8;

import android.app.Activity;
import android.os.Bundle;

public class Variables extends Activity{

        @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.variables);
}
}
```